AD-A082 074 SPERRY UNIVAC ROSEVILLE MN F/G 9/2

DATA BASE COMPUTER RESEARCH.(U)

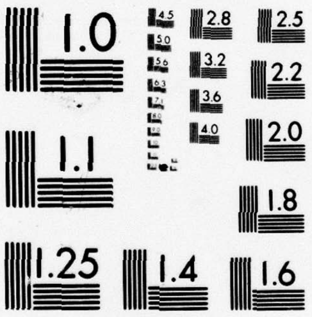
NOV 79 H A FREEMAN, J BANERJEE, R B BATMAN N00014-78-C-0487

UNCLASSIFIED TMA-00789 NL

1 OF 2

AD-

A082074



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

2. APPLICATION ANALYSIS

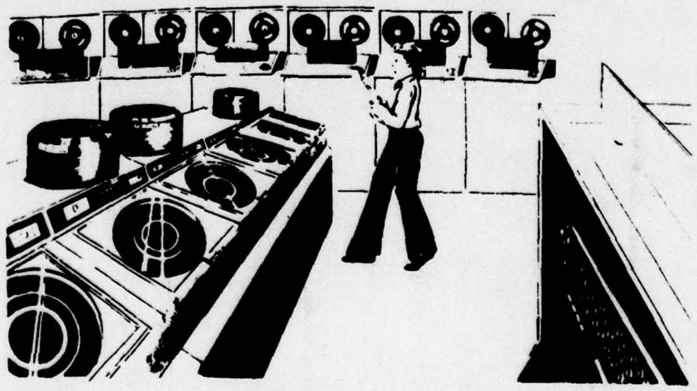
Report ONR-1

DATA BASE COMPUTER RESEARCH

Harvey A. Freeman, et al.
Sperry Univac
Roseville, Minnesota 55113

30 November 1979

Final Report for Period 1 September 1978 - 31 August 1979
Contract N00014-78-C-0487



Prepared for
Office of Naval Research
Arlington, Virginia 22217

SPERRY UNIVAC

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ~~ONR-1~~	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) DATA BASE COMPUTER RESEARCH.		5. ~~TYPE OF REPORT & PERIOD COVERED~~ (9) FINAL REP[redacted]T. 1 Sep 78-31 Aug 79,
		6. ~~PERFORMING ORG.~~ REPORT NUMBER (14) TMA-00789
7. ~~AUTHOR(s)~~ (10) HARVEY A. /FREEMAN, ~~et al~~ J./Banerjee R. B./Batman D. H./Bray H. R./Johnson		8. CONTRACT OR GRANT NUMBER(s) (15) N 00014-78-C-0487 ✓
9. PERFORMING ORGANIZATION ~~NAME AND ADDRESS~~ SPERRY UNIVAC / ~~2276 HIGHCREST DRIVE~~ ROSEVILLE, MINN[redacted] 55113		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS OFFICE OF NAVAL RESEARCH CODE 437 ARLINGTON, VIRGINIA 22217		12. ~~REPORT DATE~~ (11) 30 NOV[redacted]79
		13. ~~NUMBER OF PAGES~~ 117 (12) 119
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

UNLIMITED

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

CODASYL	DATA MODELS
COMPUTER ARCHITECTURE	PARALLEL PROCESSING
DATA BASE	RELATIONAL
DATA BASE COMPUTER	
DATA MANAGEMENT	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

DATA BASE COMPUTER RESEARCH WAS UNDERTAKEN BY SPERRY UNIVAC TO CONFIRM THE BENEFITS OF THE DBC APPROACH, AND TO DETERMINE THE ACTUAL REQUIREMENTS OF A DATA BASE COMPUTER FROM A COMPUTER MANUFACTURER'S POINT OF VIEW. DESCRIBED IN THIS REPORT ARE AN APPLICATION INVESTIGATION AND ANALYSIS EFFORT, A REVISED AND EXTENDED DATA BASE COMPUTER DESIGN, THE REQUIRED SOFTWARE STRUCTURE, A PERFORMANCE ANALYSIS, AND SOME ACCESS CONTROL AND SECURITY CONSIDERATIONS. THE RESULT IS A FRAMEWORK FOR A SPECIAL PURPOSE COMPUTER THAT WILL SIGNIFICANTLY IMPROVE THE USERS ABILITY TO MANAGE THE EVER GROWING BODY OF DATA CHARACTERISTIC OF OUR MODERN TECHNOLOGICAL SOCIETY.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE 411 660 [illegible]

Preface

This report describes the research activity at Sperry Univac to confirm the benefits of the DBC approach and to develop a comprehensive design and evaluation specification. This research, funded by the Office of Naval Research, is an extension and augmentation of an ongoing research program in data base computer systems now in progress at Sperry Univac.

The following Sperry Univac personnel performed various parts of this research and contributed to this final report:

J. Banerjee
R. B. Batman
O. H. Bray
H. A. Freeman
H. R. Johnson
J. R. Jordan
J. L. Larson
T. M. P. Lee
D. B. Russell
T. B. Wilson

The advice and guidance of Professor David K. Hsiao at the Ohio State University is greatly appreciated. Thanks also to Dr. Leonard Haynes and Marvin Denicoff at the Office of Naval Research for undertaking a unique experiment in cooperation among industry, university, and Government agency.

Harvey A. Freeman
Roseville, Minnesota
November 30, 1979

SUMMARY

Data Base Management (Software) Systems (DBMS's) have as their basic objective the improvement of an organization's control and utilization of its data resources. This goal is met by improving the availability, integrity, and security of the data base. With current computer systems, however, trade-offs must be made among the various DBMS objectives. Basically, the decision is between performance and functionality. Regarding performance, both response time and throughput are critical and must meet the users' requirements. Functionality covers availability, integrity, and security. Unfortunately, increased functionality is obtained only at the expense of performance. With data independence, for example, the DBMS translates the data automatically from its stored form to the form that the user expects. The result is that the system is easier to use, allowing the data to be restructured and modified without having to change the application programs. This automatic data translation, however, requires additional processing time and storage, thus directly affecting both response time and throughput. From another perspective, however, this performance versus functionality trade-off is simply the question of machine efficiency versus people efficiency. Considering the dramatic decreases in hardware costs as opposed to the rapid increases in personnel costs, the direction of this trade-off is apparent.

One means of using hardware to improve the performance of current Data Base Management (Software) Systems while offering increased functionality to reduce labor costs is with a data base computer (hardware) system. Using such new technologies as LSI, VLSI, microprocessors, magnetic bubble memories, and charge-coupled devices, data base computers are well-suited to the information storage and processing needs of the 1980's. In addition to offering improved performance, these special purpose computers free the general purpose host's resources for other tasks, provide a hardware assist to the security problem, and offer a more effective way of sharing data in a network.

Two different approaches to a data base computer system have been proposed and implemented to date. The first is to off-load the DBMS from the host onto a general-purpose minicomputer. Although construction of this type of system has proven feasible, the performance benefits are not realized. The main reason for this lack of performance improvement is due to the use of conventional sequential processors to perform data management functions without any hardware assists.

The second approach is to use specialized devices to perform part or all of the data management tasks. ICL has the only announced product to date in this area with their Content Addressable File Store (CAFS). ICL combined a minicomputer with special selection logic to offer high speed selection of records based on content. CAFS is an outgrowth of the three basic specialized processor approaches, CASSM, RAP, and the DBC.

Of the three basic data base computer architectures, DBC appeared the most attractive from a commercial computer company's point of view. The CASSM prototype used fixed-head disk technology that apparently will be obsolete in the 1980's although it could be adapted to any rotating storage device. However, by requiring read and write heads for every track, CASSM would be very expensive in supporting large data bases. RAP also appears viable for only very small data bases. For very large data bases, RAP may require an extensive amount of staging which may be a severe bottleneck. In addition to attempting to avoid these problems, DBC is the only data base computer that claims to support CODASYL as well as Relational DBMS's. This is an extremely attractive feature because there is no commercial Relational DBMS available as yet.

Data Base Computer Research, Contract N00014-78-C-0487, was undertaken by Sperry Univac for the Office of Naval Research in order to confirm the benefits of the DBC approach and to determine the actual requirements of a data base computer from a computer manufacturer's point of view.

The first step was an application investigation and analysis effort. A sample of the designers and users of data base applications was polled and the responses to specific questions were recorded. The application areas covered ranged from the transaction only, fast response time, simple access method airline reservations applications to the heavily batch-oriented subscriptions processing applications. The results of the survey and the subsequent analysis proved to be quite useful in establishing a set of requirements for a data base computer. It showed that DBC's proposal to cluster data in large blocks corresponding to a cylinder on a moving-head disk would be most efficient for most applications. It brought out the fact that most applications do not limit the number of people allowed to update. Finally, the survey showed that many DBMS users would have data bases in the 10 to 50 billion byte range in the 1980's. The actual questionnaire used and summary tables of the survey results are included as appendices to this report.

With the data base computer system requirements indicated by the application analysis, the DBC design was then revised and extended. The basic approach of accelerating both retrieval of the directory information and the data remained the same. Parallel transfer of large blocks of data which are then processed in parallel on a content-addressable basis is the key to this design. To this capability, interprocessor communication was added to allow for the "full" relational join and for complete sorts. Also, a solid state associative processing element was invented in order to identify unique search key and sort field values. Described in this report, the resulting design appears to offer significant performance improvement.

The Data Base Computer described in this report performs all of the functions required to support a CODASYL or Relational Data Base Management System. Preliminary analysis of the DBC indicated typical worst case performance improvements of 3 to 10 times that of an enhanced top-of-the-line mainframe. For query type operations, the performance improvement ratios might be as large as two orders of magnitude. The performance figures described in this report represent a first cut at determining performance. More accurate performance estimates will be obtained from future simulation efforts and the development of a test vehicle.

The software needed for the Data Base Computer, the connected host, and the appropriate interfaces are described in a section of this report. The objectives of the DBC software are to store and retrieve data, maintain the meta data base, service multiple requestors, and coordinate integrity and recovery across multiple DBCs. The software organization is based on a partitioned content addressable storage structure of data utilizing certain indices, large pages, and caching.

Access control and security considerations for the data base computer were investigated and are described in this report. Definite advantages from implementing data value dependent access control constraints in the Data Base Computer were found. It was also determined that all security information should be stored as part of the DBC. A general unanswered question, beyond the scope of this current effort, arose as to whether the true security decisions are made by the host computer on information it requests from the DBC, or whether all decisions are made by the DBC itself, based on little or no information from the host.

Currently, a second round of research effort is occurring. Modifications that may be necessary to support multiple DBC configurations are being investigated. Studies in the important area of data clustering with the DBC are being performed. A test vehicle is under consideration and possible implementations are under discussion. Finally, a performance model is being constructed to simulate the operation of the Data Base Computer to confirm the analytic calculations of from one to two orders of magnitude performance improvement over current DBMS's operating on general purpose computers.

Contents

Figures

Tables

1. INTRODUCTION

1.1. *Data Management Problem*

Data Base Management (Software) Systems (DBMS) development is facing a challenge to its continued growth and effectiveness. As the functionality of DBMS's increases, so does the overhead required to support them. A way must be found to reduce this overhead, increase the data management system's performance, and allow more of the computer system's processing power to be used by the application, but without reducing—indeed perhaps while expanding—the functionality of current systems.

The challenge stems from an essential incompatability between conventional computer hardware architecture and the functional requirements of data base management. Conventional hardware references data (whether stored in memory or secondary storage) by position or address, whereas most data management applications need to address the data by content or value. Significant processing and storage overhead is necessary to convert from one referenced scheme to another. The overhead can become so extreme that in some data bases the required indexes and tables are much larger than the actual data stored in the data base!

A common assumption is that DBMS functions are I/O bound and therefore could be performed on a less powerful processor, e.g., a minicomputer, more efficiently than by a large host. In many commercial applications, however, the data base operations constitute the bulk of the processing with only a few relatively simple operations on the data once it has been obtained. This is particularly true with those DBMS's using high level languages rather than record-at-a-time processing.

To meet this growing need then, new technologies such as microprocessors, VLSI, CCD and bubble memories, suggest the time is now right for a special purpose or Data Base Processor (DBP) or Data Base Computer (DBC). Using such design approaches as content addressing and parallel processing that are particularly suited to data base processing needs, large amounts of data can be addressed based on content.

There are a number of potential benefits involved in using a data base computer (hardware) system to alleviate the data management problem. First, there are various cost/performance benefits which may be realized through specialization, performance tuning, and the reduction in the need for and complexity of conversion. A second set of benefits include improved data base integrity and security. These benefits are only potential, however, since specific implementations and their trade-offs determine which benefits are realized and which ones are sacrificed for other objectives. Similarily, there are potential disadvantages and problems depending on how the data base computer is implemented. These problems include the added costs and reliability of a second machine and the potential of unbalanced resources.

Most of the work to date on a data base computer has been done in various universities. In October 1977, however, ICL "announced" CAFS (Content Addressable File Store), the first commercially available data base computer with first deliveries scheduled for this spring.[1] In the May 1978 DATAMATION, it was rumored that IBM's machine of the 80's, called the New Grad, would be a back-end and front-end system and would use System R, IBM's relational data base management system, replacing IMS.[2] Examples of the primary approaches to a data base computer that have been described in the open literature are:

CASSM	–	Context Addressable Segment Sequential Memory developed at the University of Florida.[3]
RAP	–	Relational Associative Processor at the University of Toronto.[4,5]
DBC	–	Data Base Computer at Ohio State University under Professor David K. Hsiao.[6,7,8]

1.2. Data Base Computer Research

In order to pursue the specialized computer approach to data management, Sperry Univac's Systems Research organization established a Data Base Computer Project in August 1977. The focus of this study was to determine the feasibility and cost-effectiveness of a data base computer, and to select and recommend an implementation approach. With the major trend toward increased use of high level languages, transaction processing, and distributed data bases, an application analysis is first required to determine current patterns of data base use and to project these patterns into future environments. Once these requirements are defined, then there is a basis for evaluating various hardware architectural alternatives.

An extremely attractive approach to the data base computer is that developed by Dr. David Hsiao at Ohio State University. Considerable work has been done by Dr. Hsiao and his associates over the last several years in formulating the structure and algorithms of a data base computer system, and evaluating it in the context of the major data base systems planned or in use. Augmenting Sperry Univac's data base studies, the Office of Naval Research funded Systems Research through this contract to study the implementability of a data base machine, and its ability to support practical data bases, and to validate the feasibility of the Data Base Computer Architecture design. Sperry Univac could extend and modify Dr. Hsiao's system to include those features necessary for a commercially viable product.

2. APPLICATION ANALYSIS

The application investigation and analysis effort was undertaken to provide the basis for determining the functional requirements and for evaluating the various data base computer architectures. Proposed DBC designs differ in the extent of parallelism provided, amount of clustering required for greatest efficiency, requirements for special processing of key items, and where the data is processed, directly on the mass storage devices or in an intermediate CCD or bubble buffer. Because the effectiveness of the architecture alternatives vary considerably with different application environments, it is essential to know the actual environment in which the DBC must operate. In particular, the update intensive, fast response time applications are in particular need for the type of performance improvement that may be obtained from a data base computer.

In this report, the performance and functional features of a data base computer and how they relate to various architecture alternatives are described. A critical question is how to make the trade–offs among these features—which ones represent major common user requirements and which ones are only desirable or are rarely needed. To obtain the necessary data on which to make these types of decisions, a data management questionnaire was prepared and distributed to data base users and designers. Descriptions of the survey taken, the respondees to the questionnaire, and the questionnaire itself—what items were included and the reasons for including them are also covered in this report. This is followed by a summary of the survey results and a description of the composite application environments for which the DBC should be designed. Also included are brief descriptions, characteristics, and needs of selected applications. The implications of these results on the different architecture features and alternatives are then described.

This Application Analysis portion of this report concludes with an evaluation of the survey/questionnaire approach. Also described is any additional survey work that may be desirable. New areas for investigation as well as areas in which more depth is required are included. Finally, the types of benchmarks that are needed to mirror the results obtained from the survey and the typical benchmarks that will yield these results are briefly noted.

Included in this report is the actual questionnaire (Appendix A), and summary tables of the survey results (Appendix B).

2.1. Issues and Alternatives

This section focuses on the two major issues relating data base applications to data base computer architectures. The two issues are performance and functionality. This section provides an introduction to the components in each area. The section on Usage Implications specifically discusses each of these factors in terms of the survey results.

2.1.1. Performance

In the performance area the data base computer (DBC) must provide either improved cost/performance or greater performance than was previously available at any price (e.g., the airline reservation application using the DBC back end to allow multiple hosts to access the data base or to free some of the processing power of the hosts for other tasks).

By considering current data base system loads and their growth rate, projected performance requirements can be made. These projections can then be used to determine required DBC performance.

The performance criteria must be relevant for both batch and interactive applications. Not only are there both types of data base applications, but many applications can be used in both modes. The two performance criteria used are throughput and response time. Throughput is the number of transactions or jobs processed by the system in a given period of time. Response time in an on-line system is the delay between entering the last character of the input and receiving the first character of the output. For a batch application response time is the same as the turn-around time — the period between when a job is submitted and when the output is received.

Obviously, throughput and response time are functions of the processing and I/O required by each transaction or job. Most business applications, and therefore most data base applications, are I/O bound. However, in some cases with very high transaction rates, such as airline reservations, the system may actually be compute bound, especially if the data base management system is providing considerable functionality for the application. In these cases additional throughput is possible both because certain data management functions are off-loaded from the host and because multiple hosts can now access the same data base. However, depending on the degree of parallelism that can be obtained, this improved throughput may degrade the response time.

2.1.2. Functions

A DBMS performs many functions for an application program. Therefore, one of the key decisions in defining the requirements for a DBC is to determine which of these functions should be off-loaded to the DBC and which ones should remain on the host. This section identifies six basic functions performed by a DBMS, discusses several alternatives for implementing them, and describes which application environments would favor which alternative. A later section relates these factors to the specific application environments found in the survey.

2.1.2.1. Locate

Before data can be retrieved or updated it must be located. This may require two distinct steps. First, the block or blocks which contain the desired record or records must be located. Second, these blocks must be searched to identify the specific records requested. This distinction between locate and search assumes a two step process, e.g., the use of pointers or hashing to groups of records rather than individual records, followed by a search of the records within the group.

Normally, with inverted systems there is only a single step in the locate function since there is a pointer for each key in each record. While this allows fast retrieval, it requires more storage for the pointers and more processing overhead for updates. It requires a minimum of one disk access for the set of pointers and at least one access for the data records. However, unless the records are clustered it may require one access for each data record. The same problem occurs with chaining — where pointers in each record point to the next and prior records.

With either inversion or chaining the critical question is whether the records can be clustered. If they can be, then for many applications one access can obtain all of the required records for a given transaction. In this case the two stage approach can be used. First, there is a pointer to each cluster, rather than to each record. This allows fewer, shorter pointers and therefore less storage and update overhead. Second, the individual records in the cluster are searched to determine which ones are actually needed. This can be a very effective approach, especially if the DBC architecture allows a parallel search of the records in the cluster.

Although parallelism alone can significantly improve the performance of a DBC, clustering can permit still further improvements. Therefore, an analysis of various clustering approaches is important. There are many ways to cluster data. First, records may be clustered by activity, storing the most active records in the same cluster. However, this says nothing about their joint activity. Second, a record may be clustered with other records containing the same data items, i.e., a file or record type such as department or personnel record types. Both of these approaches allow relatively unambiguous clustering. The third approach is to cluster records by the value of specific keys. This may be done across record types, as when employee records are clustered with their corresponding department record. Value clustering may also be done within a single record type where records are clustered on a single key, e.g., employee records clustered by department, skill, or location. However, a problem occurs when different applications need the records clustered by different keys. In certain cases, this problem can be circumvented by using clusters based on multiple keys, e.g., engineers located in Minneapolis, engineers located in Philadelphia, and managers located in Minneapolis and in Philadelphia. However, there is no general solution to the problem. Therefore, DBC performance improvements based on clustering are highly application dependent.

2.1.2.2. Retrieve

Retrieval is a relatively simple operation. It involves passing the selected records or the specified fields in the records to the application. (If certain fields in the record are used more frequently than others, record segmentation methods may provide significant cost savings.) However, two other characteristics of the retrieval function have more important effects on DBC architecture and performance.

The first characteristic is the number of records retrieved for a request. A DBC is more effective if it receives high level requests for sets of records, rather than requests for single records. Therefore, it is important to know whether the application generates requests for single records or for sets of records. Even in those cases where the application currently requests a a record at a time, it may be possible to convert the request to one for a set of records. For example, a series of requests for the next record may be converted to retrieve the entire set of records. The important point is that most requests for single records rather than sets are dictated by the query language rather than any inherent characteristic of the application. This is particularly true if a common data model is used to allow the DBMS to automatically convert between data models.

Also in many cases where records are being retrieved one at a time, they are clustered into files and processed sequentially by the application. Although such sequential processing does not make effective use of clustering, it does result in a highly predictable pattern of use. Therefore, the data to be retrieved next can be staged or accelerated by a DBC.

The second characteristic of the application is whether or not updates can be done concurrently with the retrieval. In many cases the data base does not have to be locked for a retrieval. However, for certain applications, e.g., generating a trial balance, locking is as important for retrieval as it is for updates. Since only the application knows whether updates will invalidate the operation, there must be two types of retrieval — one with and one without a lock. For retrievals without locks there is no problem with retrieving multiple records in parallel. However, parallel retrievals become more complicated if locking is required. In fact, some DBC architectures lose their parallelism during locking.

2.1.2.3. Update

Update is another function which the DBMS must perform. A request may result in updating either a single or multiple records. Single record updates are relatively easy since parallelism is not a factor. However, if multiple records are to be updated, there may be performance problems if parallelism cannot be provided. This will not be a major problem if most application only do single record updates.

2.1.2.4. Ordering

Frequently, when a set of records is retrieved, it must be presented to the user in a specific order. Obviously, this function can be done on the host as it is now. However, it may be possible to off-load this function to the DBC and build it into hardware. Most DBC architectures search a block of data to determine the set of records requested by the host, and then return the entire set in one message. This set may be returned in the order in which it is found or the records may be sorted and returned in the desired order. In effect there is a logical parallel to serial conversion between the DBC and the host. Therefore, it may be relatively easy to build a sort into this interface.

However, there are two limitations to this sorting. First, if the sort involves multiple keys, it becomes too complex and time consuming for some DBC architectures and perhaps should be left for the host to do. With other architectures the timing is not critical, so even multi-key sorts are possible. Second, if there are multiple blocks, e.g., multiple disk cylinders where each one is processed in parallel, the sort would be done separately on each set of records. The host would then have to merge the n (where n is the number of sets of records or blocks searched) ordered sets of selected records. However, an n-way merge is a much faster and simplier operation than the complete sort, so it may still be desirable to off-load the sort.

2.1.2.5. Security

Security involves limiting access to the data base to only authorized users and monitoring and reporting attempted violations. Most current DBMS's provide very limited data base security facilities. In the past security has not been a major concern for most users. However, this is rapidly changing. Therefore, this section discusses a range of security functions beyond those currently available.

Normally, access to the data base is controlled by passwords and user identification. A specific password may for example, give a user the authority to read several record types, to modify one of them, but not to add or delete any records, and not to access any other record type. These access controls are normally at the schema or data definition level, i.e., by knowing the record types and data items referenced in the query, the DBMS can determine if the user has the proper authorization. However, in some cases this data definition level does not provide an adequate level of security. The check must be done after the records are retrieved because it is a function of the actual data values within the record. The classic example is a medical information system. Each physician is allowed to see all of a patient's medical record, however, he is only allowed to see the records for his own patients. Thus, security checks may need to be done at two levels. First, is a particular query acceptable for a specific user? Second, even if the query is acceptable, are the specific records it retrieves permissable for the user to receive?

While the above methods attempt to limit access to authorized users, another method — encryption — attempts to conceal the meaning of the data if an unauthorized person does obtain it. Although encryption is not currently widely used, it will become much more common with the new federal encryption standard and the growing security concern of many users, especially those relying heavily on data communications. Therefore, while a DBC does not have to provide an encryption capability, it must not be incompatible with data encryption done by other parts of the system before either storing or transmitting the data.

2.1.2.6. Backup and recovery

Unlike security, backup and recovery capability has always been a major concern for data base users. A DBMS must provide the capability to restore a data base which has been wholly or partly destroyed. It must also be able to rollback the data base after an erroneous entry has been made so the correction can be made to both the original incorrect entry and to any subsequent errors which it caused. Also rollback is necessary when two transactions or programs conflict in such a way to cause a deadlock. One of the transactions or programs is rolled back and the other one is allowed to continue. All of these facilities are currently used by data base users. Therefore, any viable DBC architecture must be able to provide comparable backup and recovery facilities.

2.2. Survey

The purpose of the survey is to provide a basis for describing data base application environments to be considered in determining the functional requirements and evaluating DBC.

2.2.1. Methodology

The 42 survey questionnaires received were completed mainly by individuals who had designed, used, or had extensive knowledge of data base applications. Because this was not a statistically selected random sample, rigorous generalizations from our results are not possible. However, they do provide a better description of current data base applications for defining DBC requirements.

2.2.2. Questionnaire

The questionnaire (Appendix A) consists of three sections — Environment, Applications, and Data Base.

2.2.2.1. Environment

The environment section identifies the user's type of business, his major data base applications, and the data bases they use. It also indicates the application's share of both total system activity and data base activity. These questions allow us to rank both the user and the described application in terms of their data base activity. The characteristics and requirements of the more active users and applications is given more weight in the requirements analysis than the less active ones.

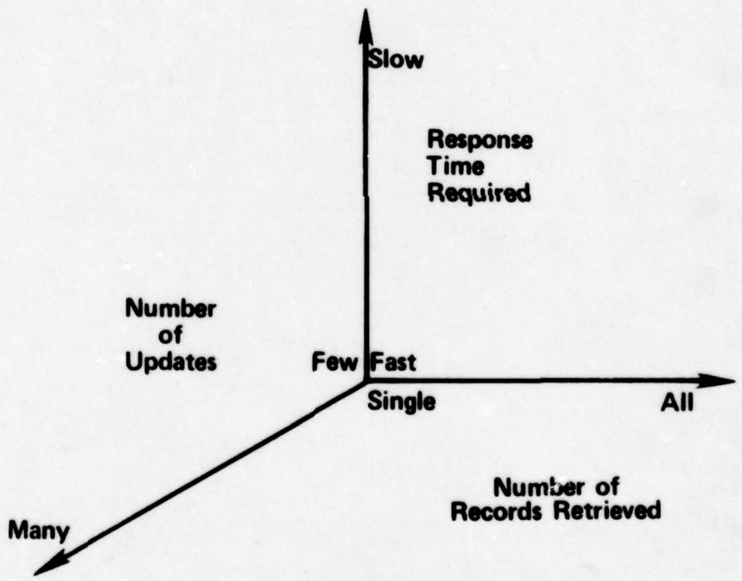
2.2.2.2. Application

The questions in this section allow us to classify an application as one of several types, each of which has different DBC requirements. If all of the search keys are known when the data base is created, then they may be processed more than the non-key fields to speed up record location, e.g., inversion or chaining. However, if the queries are unpredictable and may use any data item as a search key, then special processing is not possible, unless it is done on all of the fields which is usually excessively expensive. Therefore, the questionnaire asks if all of the search keys are known in advance.

Assuming the search keys can be identified, a trade-off must be made between using a key for faster location and retrieval and the added overhead required when one of the key fields is updated. Since the overhead is only paid when the search key is modified, there is also a question to determine how frequently this occurs.

Any query or application whether batch or transaction can be classified along the three dimensions shown in Figure 2-1.[9] Two of the dimensions involve the number of records retrieved or updated. This number or the size of the response set can range from zero to all of the records of a particular type. However, the actual number is not as important as the relative number. Therefore, the possible responses to both questions are one, few, many, or all. Although the user may be retrieving or updating a single record, several additional records may need to be retrieved to select the appropriate record. This is partly determined by asking the number of levels that must be traversed to obtain the selected record. The third dimension is response time. Interactive processing requires a faster response time than batch applications, where response time is really the same as turn-around time. Response times are also requested for both retrieval and update regardless of whether the application is used in batch or transaction mode or a combination of modes.

Figure 2-1. APPLICATION TYPES



Frequently, when a record is retrieved, only a few of the fields in the record are needed. If this is the case, record segmentation techniques can provide major costs and performance savings. Therefore, it is important to know how much of the record is needed.

Finally, concurrent updates create a problem for any data base system, since locking mechanisms, deadlock prevention, and roll back methods are needed. The questionnaire determines both the number of concurrent users and the number of them that can simultaneously update the data base.

2.2.2.3. Data base

The final section of the questionnaire focuses on the data base, independent of the application usage being described. It asks for the size and growth rate of the data base, the number of fixed and variable length record types, and the overall minimum, maximum, and average record size. This data indicates the size of the data base that the data base computer must be able to support. Also variable length records require more overhead to support.

The questionnaire asks how the various record types are accessed. These access methods and the data structure (hierarchical or network) provide information about the complexity of the data base and the overhead required to locate records. Related to this complexity, there are also questions about the total number of keys in the data base and the volatility or frequency with which the key values are changed. The number of keys provides an indication of the amount of storage required if inversion or chaining is to be used for accessing the data base on these keys. The volatility of the keys indicates how frequently overhead will be incurred to reorder the file or update and reorder the corresponding pointer tables.

2.3. Survey Results

The questionnaires that were returned as part of our survey, covered a wide range of application areas. First there are the transaction only, fast response time, simple access method applications such as airline reservations and on-line banking. The next group comprised the mixed batch and transaction, usually medium demand and complexity such as manufacturing, student personnel records, chemical sales, and telephone directory assistance applications. The third set were the heavily batch oriented such as subscriptions and state elections.

2.3.1. Applications and Requirements

Two application areas which are representative of the diversity of data mangement tasks are described in the following subsections.

2.3.1.1. Airlines

The airlines processing applications pose some very serious problems for even the fastest computer systems. Experiencing a growth factor of 8 to 15% per year, the airlines find themselves, for the most part, to be compute bound. Consider the following typical requirements for the 1980/1981 period:

* Process 175 to 200 transactions per second

* Access an integrated data base:

- of 4 to 7 billion bytes (duplexed total)
- 2400 to 2800 accesses per second
- Approx. 1000 bytes transferred per access

* Execute 17000 user application instructions per transaction
* Accommodate a communications network of 5000 to 8000 terminals
* Provide multi-processing, multi-programming, multi-application environment
* Provide 99+ % availability, 24 hours x 7 days
* Enable physical partitioning of all redundant components for security purposes
* Minimize cost of redundancy
* Perform above in degraded mode

Whereas today's (1979) operations have the following features:

* 23000 operating system instructions per transaction
* 40000 instructions per transaction
* 117 transactions per second
* Severely access limited mass storage
* Limited capability for large networks
* 50 hour MTBS

The characteristics of an average transaction are:

Instruction executions:		
17000 user application instructions		43%
23000 operating system instructions		57%
	% of 40000	% of 23000
Operating System Instructions:		
11680 - Mass Storage I/O	29%	51%
3000 - Communications	7%	13%
5460 - Other Operating System Overhead	14%	24%

2860 - Misc.	7%	12%

Mass Storage I/O:	Logical Accesses	Physical Accesses
Read	4	4
Duplex Write	4	8
Total	8	12

Approx. 1000 bytes transferred per avg. access

Communications:

50 characters in	(text only)
200 characters out	(text only)

Central Site Existence Time:

600 milliseconds or less - average

(from last input character received to first output character ready to transmit.)

Transaction rate (per second):	Average	Yearly Peak Hour
0 Small airline	3	6
0 Medium airline	15	22
0 Medium-large	25	40
0 Large airline	80	160

At present, commercial airlines meet their application needs with computer programs that handle:

Seat reservations,	Flight planning,
Message Switching,	Planning of crew duties,
Passenger check-in,	Engine maintenance,
Weight and balance,	Hotel bookings, and
Fare quotation,	Cargo control.

Due to the number of transactions per second, the response time requirements and the overhead involved, most airlines application programs do not use a DBMS at present. The Reservations application uses a relatively simple access procedure but requires very fast response time. The average reservation transaction makes 5 data base reads and 3 writes. The communications network adds the approximately 2 seconds to the 3 second response time. This being the case, the complex query handling and relational aspects proposed for data base computers might not even be needed for the standard type of reservation function.

What definitely would help would be simply off-loading the data base functions from the host. Then even if the access took the same amount of time on the DBC as the host, the benefit of freeing more of the host's resources for application processing could be obtained.

Performance improvement could be obtained if the number of I/O accesses could be reduced. Because only one search key is usually used in the Reservation application, the potential for clustering exists. Larger blocks of information (the contents of a cylinder or a cache) could be used to reduce the likelihood of subsequent disk accesses.

2.3.1.2. Manufacturing

The Sperry Univac UNIS Industrial System is a comprehensive production and inventory control system designed especially for manufacturing companies. Major features include inventory management, work order management, planning and scheduling, and master data processing.

UNIS provides the user with a comprehensive data base controlled through DMS 1100 or DMS 90. Some of the UNIS programs are quite complex using several levels of functions in a hierarchical structure appropriate to problems, such as material requirements planning, which require extensive amounts of data (100% of the data base). Others are simpler in design but use many UNIS functions in solving many problems serially, such as updating the data base with such input as order releases, allocations, purchase receipts, stockroom issues, and the entire range of inventory accounting transactions.

The largest UNIS users, executing on 1100 Series machines, have a centralized data base that may contain as many as 100,000 parts and 300,000 structures. Just the initial load of the data base may take 50 hours. Series 90 users employ a distributed data base for their UNIS operations. Also, users are encouraged to add such options as Accounts Receivable on top of UNIS. In many cases, users may even double the size of the data base by adding these related applications.

The data base for UNIS is defined by its schema to consist of records arranged in two groups. The larger group includes data records while the smaller group is composed of records used for tables containing constants and transaction processing routing instructions. All "owner" records are directly accessible through their logical keys or through sequential searches. Each owner record has only one key through which direct access is obtained. For the member records, there are at most two keys or paths through which access is made.

The questionnaires received on UNIS applications indicate 85% of the requests in a transaction environment use one search key while 15% use two. The main access paths are customer order number and part number. Records are currently organized around one or the other number. This makes for efficient processing by customer order number, for example, but not by part number. The potential appears to exist, however, for forming the products of the sets of these two numbers and "clustering" or organizing the records around these products. Then this type of operation would lend itself to Hsiao's Data Base Computer scheme.

The sequence of processing in UNIS is very important. Time or topological sequencing inhibits activity scheduling until all previous activities are processed. This absolute requirement for processing in sequence may cause various difficulties for the parallel processing portions of data base computers.

The Automated Parts Inventory (API) application used by Sperry Univac's Roseville manufacturing operations is presently a non-DMS batch operation. Access is index sequential through a part number to one fixed length record type. Currently, this application is being converted to UNIS.

A feature that would be desirable for its projected use in a transaction environment is an efficient count mechanism. If, for example, the number of items that meet the criteria for a query were made known to the user, he or she would know if the query were too general, and could respond with a more specific query. The parallel searching mechanisms that are an integral part of a data base computer system would be very useful in this regard. Similarly, this feature could be used to estimate the magnitude of a job based on the number of record occurrences and thereby allow efficient job scheduling.

2.3.2. Summary of Results

This section includes a summary, highlights, and notes on the various questionnaires received. An item by item summary for each question, including minimum, mean, and maximum value, can be found in Appendix B.

2.3.2.1. Environment

Although the questionnaire distribution procedures did not guarantee a statistically random sample, the desired range of applications were covered. Small applications such as airline message switching or manufacturing information search and retrieval as well as large ones such as airline reservations and on-line banking were covered.

Most applications were not dedicated to a particular system so that performance increases for these applications would allow more resources to be available for the othe- applications on the system.

2.3.2.2. Application usage and description

Fourteen percent (14%) of the applications described were transaction only, 14% were batch only, and the remaining 72% were both batch and transaction in varying degrees. Most retrievals in a transaction environment were for a few (1-5) records with a response time in seconds. Batch requirements were generally overnight with large number of records processed.

Response to the updating questions were usually similar to those for retrieval. Most people indicated that the response time required for updating was the same for retrieval for the more than 50% of the transactions requiring updates. There did not appear to be any restrictions on who could update. In all of the questionnaires received, the number of concurrent users allowed to update was the same as the number of concurrent users. Survey results also indicated correspondence between response time required and number of transactions per hour. The higher the number of transactions per hour, the faster the designated response time. Given the response time required and the lack of restrictions on the number of users allowed to update, care must be taken to insure that deadlock is prevented, other users are kept from changing a record or file when an update is in progress, and roll back techniques restore the file to the same position for all users.

Most of the applications used two or fewer search keys which were identified in advance thus allowing special processing techniques, such as inversion, to speed up record location. Batch applications were less likely than transaction ones to identify all of the search keys in advance. Because the response time requirements for batch are much slower than for transactions, time would be available to search all of the not-identified-as-key fields. Thus, incorporating into the data base computer techniques that assume all of the key fields were identified in advance would in general result in faster processing.

Only two respondees indicated that the application being described most frequently required only one field, the remainder required multiple or all fields. This would indicate that record segmentation techniques would not be that useful.

Finally, some of the answers to the question on the number of levels or record types traversed to obtain selected records had to be disregarded. A few people appear to have answered this question by stating the numbers of records they had to look for before obtaining the desired record as opposed to the number of different types of records they had to use.

2.3.2.3. Data base usage and description

Once the data base is established, the growth rate is usually less than 2.5% per month. The smallest data bases in the survey showed the highest growth rates.

In the majority of cases, only fixed length record types averaging 347 bytes per record were used. Most of these record types were accessed via another record type or randomly, through a calculated or hash value or directly from the key value. Only one small data base was organized sequentially. Of the remainder 60% were network types and 40% were hierarchically organized.

Considering all of the record types, its average number of search keys was 2.6. This corresponded to the application usage of 3 or fewer search keys. This implies that there usually were not more than 1 application running against the data base.

2.4. Usage Implications

The three primary functions are locate, retrieve, and update. These functions may be done on ordered or unordered files, but with significant performance differences. However, a file can be ordered on only one key, although within that key there may be secondary keys.

If a file is being processed sequentially, it should be ordered on the appropriate key. If random retrievals are then made against this file, there are several ways to locate the record. A unique key or identifier can be found in n/2 compares, regardless of whether that key value is in the file or not. Using a binary search, it will only take n log n compares. For retrieval only these location methods are relatively effective. However, updating an ordered file can be difficult and time consuming. New records must be added on the proper position and the rest of the file shifted. Also if a key is changed the record must be moved to a new position. This delay creates problems because the survey indicates many updates are being done and that they required a response time comparable to retrieval.

An alternative is to use unordered files. Although normally it takes longer to locate a record in an unordered file than in an ordered one, this is not necessarily the case with a DBC. First, the file can be segmented and the segments processed in parallel. Second, records can be clustered by activity rather than by key value, so the most active records can be searched first. Therefore, by using sufficiant parallelism, any arbitrary location speed is possible. Also with an unordered file there can be many keys rather than just one as with an ordered file. The main benefit of an unodered file is that updates can be done much faster – an important requirement found on the survey. New records can simply be appended to the end of the file. Also a record does not have to be moved when its key is changed. This means that update response times are not significantly longer than retrieval times.

Regardless of whether the file is ordered or not, certain access control are needed to prevent conurrent updates of the some data. Because the survey shows that 50% of the transactions require updates and that in general there can be any number of concurrent users, access controls are necessary and they must operate very quickly to avoid delaying succeeding transactions.

In most cases only a few records are retrieved. These records may then need to be ordered, so this capability must be provided. However, since large files are not constantly being sorted, this function can be implemented on either the DBC or the host. The sort does not have to optimized for large files.

2.5. Application Analysis Conclusions

2.5.1. Application Analysis Approach

The survey approach to obtaining information on the actual environment in which the data base computer must operate was quite useful. It provided information necessary for determining the functional requirements and for evaluating various proposed architectures of a data base computer. It confirmed such intuitive feelings such as clustering data in a large block, such as a cylinder on a disk, would be most efficient for the majority of applications. It highlighted the fact that most applications do not place restrictions on the number of people allowed to update. Also, representative applications were identified and data obtained that can be used to characterize a set of standard benchmarks that will allow comparisons between the current approach and one using a data base computer.

Although the survey identified current data management applications and techniques, it does not necessarily indicate what will be done in the future. What would the users do if they had additional features, techniques, hardware, etc., available to them? What nev' applications would then be designed? Subjective questions on the future use of a data management system were not included in the questionnaire in order to obtain information on current usage as quickly as possible. Questions involving projections of current applications form part of the additional analysis requirements described in the next subsection.

2.5.2. Additional Analysis

The survey and analysis to date should be just the first step in an iterative procedure for characterizing applications, their data management system usage, and requirements. The second step is to ask more specific questions as well as to expand the scope of the questions.

In-depth probes could be made to determine, for example, the total number of record types in a network, the average and maximum number of record occurrences per record type, the search keys for each type. And the frequency of reference to each record type. Questions regarding back-up and recovery could be added. These were not included in the current questionnaire because the impact of back-up and recovery on various data base computer architectures is not known at present and therefore, the types of questions to ask were not clear. Expansion of the questionnaire to indicate more in the way of future needs and activity should also be included.

The next step in a survey approach is to conduct interviews with selected respondees. This process enables features and requirements unique to a specific application to be more easily identified. It also allows for more of a feeling for the application and is helpful in gauging the validity of the questionnaire responses. It is also helpful in discovering items that were overlooked when formulating the questionnaire. Both the application users and the designers should be included in this phase.

The data gathered through these steps can be used in constructing various benchmarks and simulations and in evaluating data base computer designs. As more information or more statistically accurate information is needed, the process should be repeated using a wider sample.

2.5.3. Benchmarks and Simulations

The ability to compare current data management techniques and performance with proposed alternatives is essential to a data base computer study. One means of achieving this is through a set of benchmarks that will yield operating characteristics and statistics on current products.

At least three data management benchmarks, corresponding to the transaction only, batch only, and mixed transaction/batch environments, should be obtained. Characteristics of these benchmarks should be compared to the results of the data management survey. Performance statistics then obtained can be evaluated in light of the average and extreme representative applications in each general category.

The data generated through this survey and analysis can also be used in simulations to evaluate data base computer designs. Used in conjunction with the benchmarks, a picture of the impact of a data base computer upon current systems and applications can be obtained.

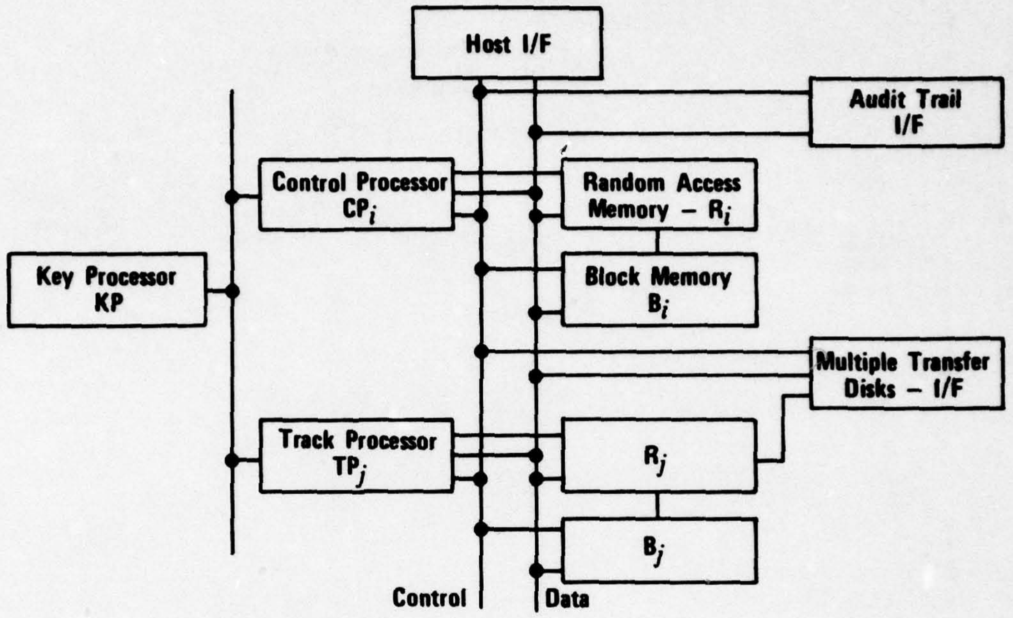
3. DATA BASE COMPUTER DESIGN

With the data base computer system requirements formulated, the DBC design was then revised and extended. The basic approach of accelerating both retrieval of the directory information and the data remained the same. Parallel transfer of large blocks of data which are then processed in parallel on a content-addressable basis is the key to this design. To this capability, interprocessor communication was added to allow for the "full" relational join and for complete sorts. Also, a solid state associative processing element was invented in order to identify unique search key and sort field values. Described in the following section, the resulting design appears to offer significant performance improvement at a relatively modest cost.

3.1. Architecture

The architecture of the extended Data Base Computer is shown in Figure 3-1. In the original DBC design, two loops were used for processing commands or queries. The structure loop identified the cylinders containing the desired records, performed preliminary security checks, and clustered records to be inserted in the data base. The data loop elements accessed and stored the data base and post-processed retrieved records. Since the functionality of both the data and structure loops utilizes the same parallel processing techniques, this design incorporates both of these functions into a single structure.

Figure 3-1. DATA BASE COMPUTER ARCHITECTURE



The Data Base Computer design has five major types of components:

1. Control Processors (CP's),
2. Random Access Memory Modules (R's),
3. Block Memory Modules (B's),
4. Track Processors (TP's), and
5. Key Processor (KP).

These components, their interconnection, and interfaces to externally connected devices are described in the following paragraphs.

Implemented in the form of microprocessors or VLSI logic chips Control Processors accept commands from the host. Depending on the software interface, they are either data manipulation-level commands from a CODASYL based DBMS or high-level query type commands of a relational nature such as Sequel[10] or QLP 1100[11] commands. The commands are then processed by the CP's and an appropriate set of parameters and commands is generated for TP, KP, audit trail, and/or disk operations.

The data base as well as directory information that cannot be kept resident exists on a series of multiple or parallel transfer disks. Parallel transfer from a single disk unit at one time is prefered for performance and reliability reasons. However, since the basis of the DBC operation is to process multiple tracks in parallel, these tracks could be obtained from multiple units. The disk controller interface handles the transfer of information from the drives to the Random Access Memories directly or to the Block Memories indirectly through the data bus. The usual error correction, defect processing, and other features commonly found on disk controllers today are provided in the interface. The ability to provide parallel transfer of data from a number of tracks on a disk unit does not involve any technological breakthroughs. Ampex Corp. has modified one of their 9300 series 300 megabyte disks to offer the transfer of up to 9 disk tracks in parallel.[12]

The Random Access Memory Modules in the present design contain:

1. track processors' programs and local variables,

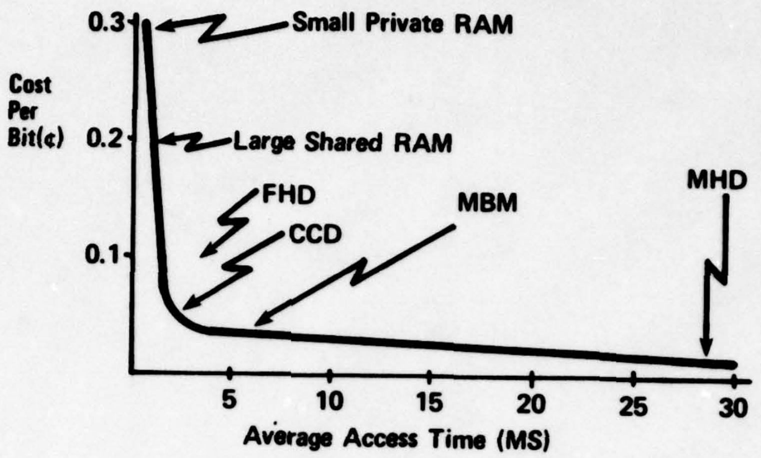
2. system tables, and

3. buffers for information being transferred to and from the mass storage devices.

The exact amount of random access storage has not as yet been determined and will probably depend on the results of a performance simulation study. The programs executed by the TP's and CP's are basically simple and should not require an unreasonable amount of storage. Paging of this code, however, is unacceptable and, therefore, the R's should be sized to always have this program code resident. System table information, on the other hand, can be paged. The advantage of using large blocks of information (for example, a cylinder's worth) instead of the present 112 or 448 word blocks is that the directory information is greatly reduced. Therefore, the dominant factor in sizing the R's will be the track buffering requirements. This in turn is dependent upon the relative speeds of the TP's and the disks and the presence of secondary storage areas. Parameters of current microprocessors and a 1.2 Mbps per track disk rate (Ampex Parallel Transfer Disk PTD-930X) were used to estimate that three tracks' worth of buffer storage (60 KB) would be sufficient for reasonable throughput provided secondary storage is also present on the DBC. A reasonable size, then, for the Random Access Memory Modules might be 96 KB.

Since many of the operations of the DBC do not require random access to the data, the secondary storage area (Block Memory) provided on the DBC can be implemented with serial storage devices. Magnetic bubble memories, charge-coupled devices, and even fixed head disks all offer less expensive storage than RAM at much higher transfer rates than moving head disks (see Figure 3-2). This Block Memory is used to store overflow data from the R's, selected records being held for transmittal to the host or disks, and modified data that the user has already committed. Although the technology that will be the most cost-effective in the 1980's is still in doubt, the non-volatility of bubble memories make that technology particularly attractive. There is also the possibility that the cost of RAM's will continue to decline at such a rate that this, in fact, would be the most cost-effective. If RAM is indeed used, the 2 MB per module of storage estimated for the Block Memory can be reduced.

Figure 3-2. STORAGE TECHNOLOGIES



In general, information from the appropriate disk tracks is transferred simultaneously (i.e., in one disk rotation time) into the associated Random Access Memory Modules. *A'ter the R's have been loaded* with a track's worth of data, the Track Processors start operating on the data asynchronously performing the required functions. These functions are assigned by the Control Processors via a set of task queues in the TP's corresponding Random Access Memory and are described in the naxt subsection. The microprocessor chosen to implement the TP's should be the same as those for the CP's to provide a graceful degradation capability.

The Key Processor (KP) is an unique element that is used to accelerate certain data base operations by providing a temporary partitioning of a file over an attribute value space. As shown in Figure 3-3, the Key Processor consists of five major components:

1. Control logic,

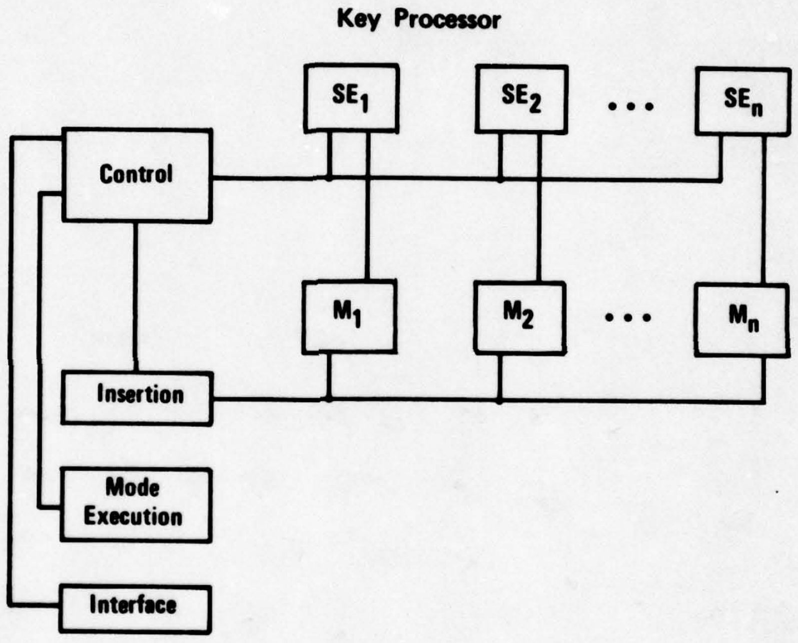
2. Search Elements (SE's),

3. Memory Modules (M's),

4. Insertion logic, and

5. Interface logic.

Figure 3-3. KEY PROCESSOR



As shown, the Key Processor has been designed to use conventional RAM chips. At the present time, however, the possibility of using custom designed chips is being investigated. One possibility would be to format 16K chips internally into groups of 64 words of 256 bits each. Then for reasonably sized storage (2000 arguments), 32 chips would be necessary. In addition to the storage on each of these chips, all of the search logic required for operation would be included.

Four modes of operation are programmed into KP in order to perform the required task. These include:

1. Insert: The KP returns an "order of arrival" or index number, k, to the requesting TP if the presented argument, X, is already in KP. If the argument is not there,

it will be inserted and a value of k assigned and returned to the requesting TP.

2. Match: The KP determines whether or not the presented argument is already in KP. If present, a non-zero value of k is returned.

3. Mark: A bit in the KP corresponding to the presented argument, X, is set.

4. Retrieve: Two types of retrieval are possible. In the first case, an unmarked argument, Y, is retrieved and its corresponding mark bit set. In the second case, a marked argument, Z, is retrieved and its corresponding mark bit reset.

The argument string, X, that is presented to KP in the first three operational modes is also accompanied by a count threshold, C_t. This number, C_t, is used to determine when a sufficient number of records have accumulated in the random access memory to warrant transfer to the block memory. The use of these operations in functions performed by DBC is given in the next subsection.

Although corrupted data will always occur due to user or program errors, the effects of hardware failures can be reduced through the use of fault tolerant techniques. Since DBC is constructed from sets of identical hardware modules, additional modules can be added as spares and switched into operation when a failure in an existing module is detected. To ensure that there is no single point of failure in the system, the interface logic and the interconnecting buses would all have to be replicated. The only item in the current design that would not need replication is the Key Processor. If KP fails, the Control Processors, with the appropriate software, can perform the required KP functions, although at a slower rate.

3.2. DBC Operation

In general, all of the components in DBC operate synergistically to perform the functions described in this subsection.

3.2.1. Search

A cylinder slice at a time is loaded from a particular parallel tra or from a number of conventional disks into the shared memory modules. Initiated by a CP, all of the TP'S search their respective R's for the records that meet the search criteria. Using microprocessors for TP's and buffering the information allows for search arguments of arbitrary boolean complexity. The records that are found that qualify are either sent to the host or moved to the Block Memory (B's) to await further processing.

3.2.2. Projection

This operation involves selecting only a portion of the qualifying records which, depending on the uniqueness of the remaining information, may result in duplicate records. If Projection with the elimination of duplicates is desired, the Key Processor in Figure 3-3 is involved. In this case, the argument string of the qualifying record is presented to KP which is operating in the Insert mode. If the argument string is already in KP, the record is a duplicate and discarded. Otherwise, the new string is stored in KP and the record is prepared for output to the host or disk or moved to Block Memory to await further processing.

3.2.3. Full Join

In the Full Join, a record from a relation (or file) A is concatenated with a record from a relation (or file) B if they have the same value of a common attribute. The Full Join is performed in several steps. First, for each record in A, the particular TP transmits the argument string to the Key Processor which is operating in the Insert mode. When a number, k, is returned by KP, the record is then added to list number k in the appropriate random access memory. Second, after all of the records in A have been treated, the records in B are processed. In this step, each of the argument strings from the records in B are sent to KP which is now operating in the Match mode. If the particular argument string of the record is not in KP, the record is discarded. Otherwise, the list number, k, containing the appropriate records to be concatenated, is reported. The requesting TP then concatenates ("joins") the record from B with all of the records from A in its memory on list k to form a new set of records A' and then outputs them. When all of the records from B have been processed in this manner, the Full Join operation has been completed.

3.2.4. Implicit Join

The Implicit or Half Join selects records from a relation (or file) A that have a common domain with records in relation (or file) B. No concatenation is required and, therefore, the Implicit Join is a simpler operation than the Full Join. During the first phase of the Implicit Join, the unique argument strings of file B are stored in the Key Processor. In the second phase, argument strings or values from file A are checked by KP operating in the Match mode. All matches with values in KP will result in the output of records from A. Records from A that do not have a corresponding value in the KP are simply discarded.

3.2.5. Modification

In order to change existing records or tuples, the keys of the requested records to be changed are first inserted into the KP (Insert Mode). Then the cylinder slice of records to be changed is read from disk or block storage into RAM. The keys of the records in this cylinder slice are then checked against those in KP (Mark mode). Finally, with KP in the Retrieve mode, the marked items are obtained by TP's and the appropriate changes made. Items in KP that are left unmarked are either requests to change records that are not in the retrieval cylinder slice or are requests to change non-existant records. In the former case, additional cylinder slices must be obtained and processed. In the latter case, the user is notified of the error condition.

3.2.6. Deletion

This operation is similiar to that for Modification with the records corresponding to the marked items being deleted.

3.2.7. Addition

The first two steps of adding a record to a file are similiar to those for Modification or Deletion. In the third step, however, the unmarked items are obtained and the appropriate records added. Items that are left marked in KP indicate that a record corresponding to the requested addition already exists and inserting this record into the file would result in a duplication.

3.2.8. Sort

Given the parallel processing elements in the DBC, many types of parallel sorts are possible. One example is where each TP first does a standard sort/merge of the contents of its own portion of the shared memory, R. This is followed by a merge of the contents of the lower portion of this memory with the upper portion of its adjacent TP's memory. These two operations are alternated until the file is completely sorted. In any case, the performance improvement with a parallel sort should at least be on the order of n, the number of TP's, times that of a single sequential sort.

More complex operations such as Divide or Set Intersection, involving multiple Joins or Projections, are also possible. In performing joins or projections, the situation may arise where the memory space in the Key Processor becomes full. If this should occur, each TP continues to process its segment until the end but will place each record that does not fit in KP on an overflow list. When all the TP's have completed this operation, the KP's memory space is cleared and the overflow records are processed. For those cases where the key value or attribute string is too large to fit into KP, the value is hashed to a value that will fit. KP in this case, is put in the Insert mode and the number is reported back to the requestor. The actual value is then placed on the list corresponding to the number that KP returns. In subsequent operations when the actual value is needed, the appropriate list is traversed to obtain this value. Thus, Key Processor can also be used as a hashing aid.

4. DATA BASE COMPUTER SOFTWARE

4.1. Software Overview

The DBC software and associated host software consist of a network of software subsystems that communicate with each other via protocols that include commands, data, and responses. Thus it is called a software subsystem network.

A subsystem in this software network has the following characteristics:

- an external interface that defines the commands, data, and responses which it will process (i.e. an architectural interface);
- internal processing logic which is local and subject to change so long as the external interface remains intact;
- internal data structures which are local and subject to change so long as the external interface remains intact;
- requirements on the Meta Data Base (one of the many inputs used to develop the Meta Data Base);
- requirements on other subsystems that it may call to perform functions for it.

A particular subsystem may itself be partitioned internally into nested subsystems.

The top-level software subsystem network is shown in Figure 4-1 and consists of the following major components:

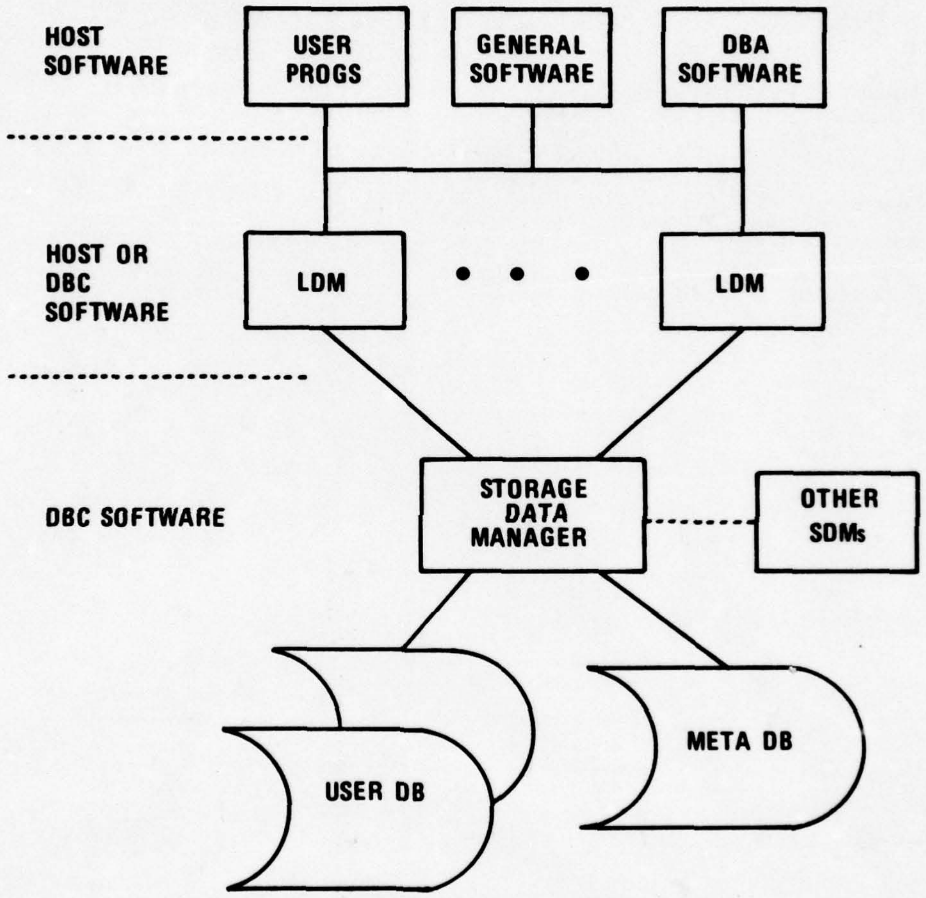


Figure 4-1. SOFTWARE OVERVIEW

4.1.1. Host Software

- User Programs — These are the user's application programs that issue data manipulation language and file I/O commands to the logical data managers. They process the user applications using the data management software for the storage and retrieval of data.
- General Software — This category of software includes the compilers, utilities and executive functions which utilize the DBC for storage and retrieval of data.
- Data Base Administration Software — This category of software includes the data definition processors of the various logical data models, file administration software for migration and archiving, utilities for data base verification, data base design aids, data dictionary processors, etc..

4.1.2. Logical Data Managers (LDMs)

The logical data managers are those functional systems that support particular logical data models. Any logical data model could be supported in this way and the following logical data models are currently being considered: Network (Codasyl, DMS 1100), MISAM, Direct, Relational, OS 1100 primary I/O. Other data models may be considered in the future.

The logical data manager presents a logical view of the data to the user program. It transforms the logical DML commands into data storage language (DSL) commands to the Storage Data Manager which operates in the DBC.

The DSL command may be a record level command, a command related to an aggregate of records, or a query. Based on the data model and the existence of subschemas the LDM may convert or map the data before delivering it to the user program. The LDM may request an aggregate or cluster of records from the Data Storage Manager and handle them internally.

4.1.3. Storage Data Manager (SDM)

The Storage Data Manager (SDM) is the software subsystem operating on the Data Base Computer. It is responsible for the storage and retrieval of data as well as other data management control functions.

The SDM includes the necessary control program and function routines to accomplish:

- Storage and retrieval,
- Space management,
- Meta Data Base maintenance,
- Device control,
- Access Control and Security,
- Integrity,
- Accounting,

- Recovery.

There are one or more storage schemas which the SDM manages. Where data bases spread to multiple DBCs, the multiple SDMs interface according to defined protocols to provide distributed data management control for integrity and recovery.

4.2. Meta Data Base Overview

The Meta Data Base is that group of data objects needed by the entire Data Management Complex in the performance of its function. It is a true data base in that it stores the information in defined data structures and is utilized by many programs or applications (e.g. compilers, DBA routines, LDMs, SDM, etc.).

In general the Meta Data Base contains information defining the data objects of user data bases (items, sets, relations, files, etc.) and directories which contain information on occurrences of these data objects and their activity (area directory, activity information, physical indices, etc.).

A schematic of the main components of the Meta Data Base is shown in Figure 4-2.

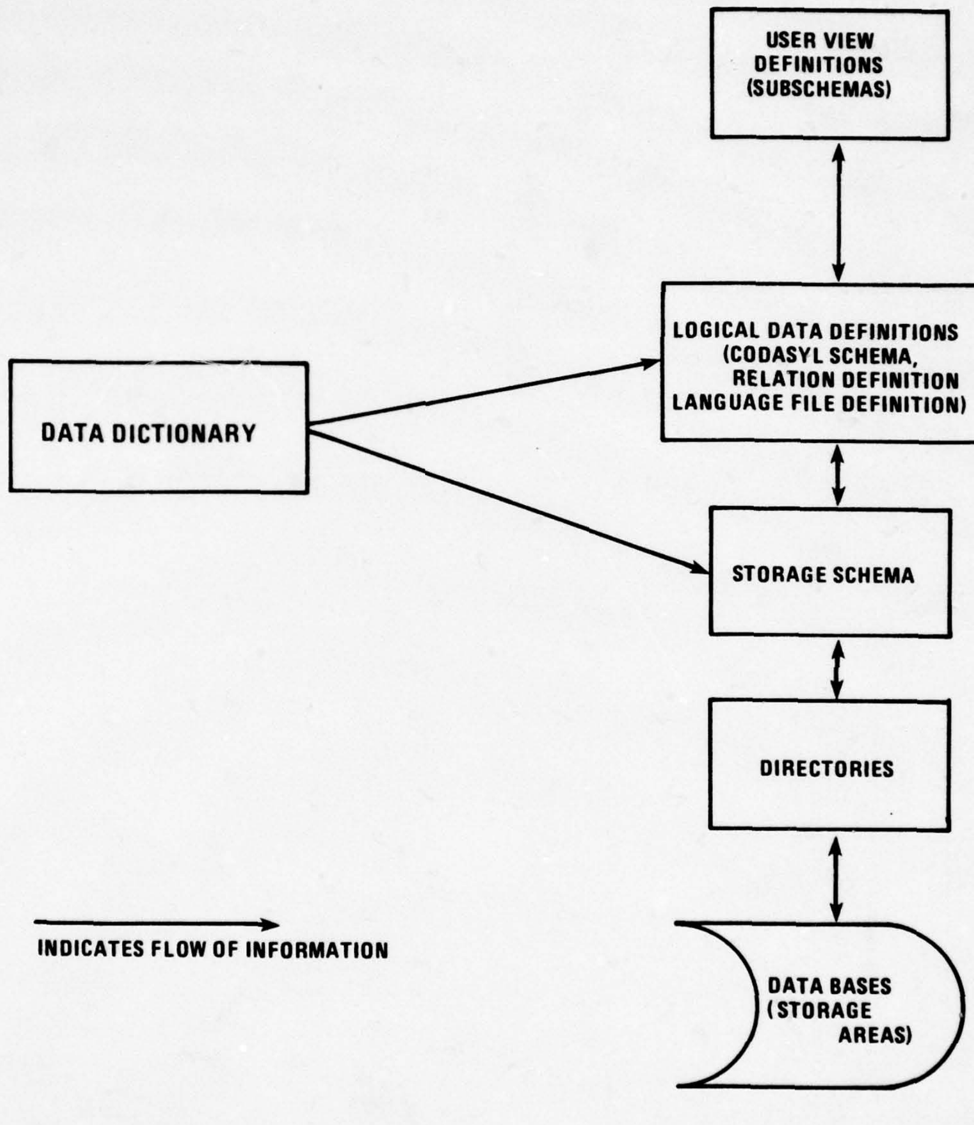

USER VIEW DEFINITIONS (SUBSCHEMAS)
LOGICAL DATA DEFINITIONS (CODASYL SCHEMA, RELATION DEFINITION LANGUAGE FILE DEFINITION)
DATA DICTIONARY
STORAGE SCHEMA
DIRECTORIES
DATA BASES (STORAGE AREAS)
INDICATES FLOW OF INFORMATION

Figure 4–2. META DATA BASE SCHEMATIC

4.2.1. Data Dictionary

This is the basic information set from which other definitions and schemas are developed. The data dictionary contains an inventory of the basic data objects from which records, sets, files and data bases are contructed. The entities defined with it are of three types:

Data Domain –	A named abstract data type with a specific set of possible values, a representation form, and a narrative explanation of its meaning;
Data Item –	A named data object whose possible values are taken from a specific domain identified by name, together with a narrative explanation of its meaning and use;
Group Item –	A named group of data items, whose meaning and use is determined as a group.

Cross reference information is maintained in the Data Dictionary showing where data domains are referenced and where data and group items are used by the various logical schemas and storage schemas.

The Data dictionary is utilized primarily by DBA routines when creating Logical Data Definitions and Storage Schemas.

4.2.2. Logical Data Definitions

This information set contains the CODASYL Schemas, Relational Schemas, and conventional file definitions. These are the logical data definitions pertaining to the various data models. If new data models are developed, their logical data definitions would be included. The Logical Data Definitions are stored and retrieved by the SDM on behalf of the logical data managers who are their primary users.

4.2.3. Subschemas

Subschemas define the external view of data as a particular user program wants to use it.

4.2.4. *Storage Schema*

The storage schema defines the techniques to be applied to the placement of records, the expansion of areas, logical to storage record mapping, and index definitions.

4.2.5. Directories

The directories contain current information on the occurrences of storage areas. In here is maintained a catalog of storage areas, where they are currently stored, physical indices if appropriate and activity records.

4.3. Software Subsystems and Intercommunications

This section deals with a further breakdown of the software subsystems (see Figure 4-3) and identification of the major intercommunications interfaces (see Figure 4-4).

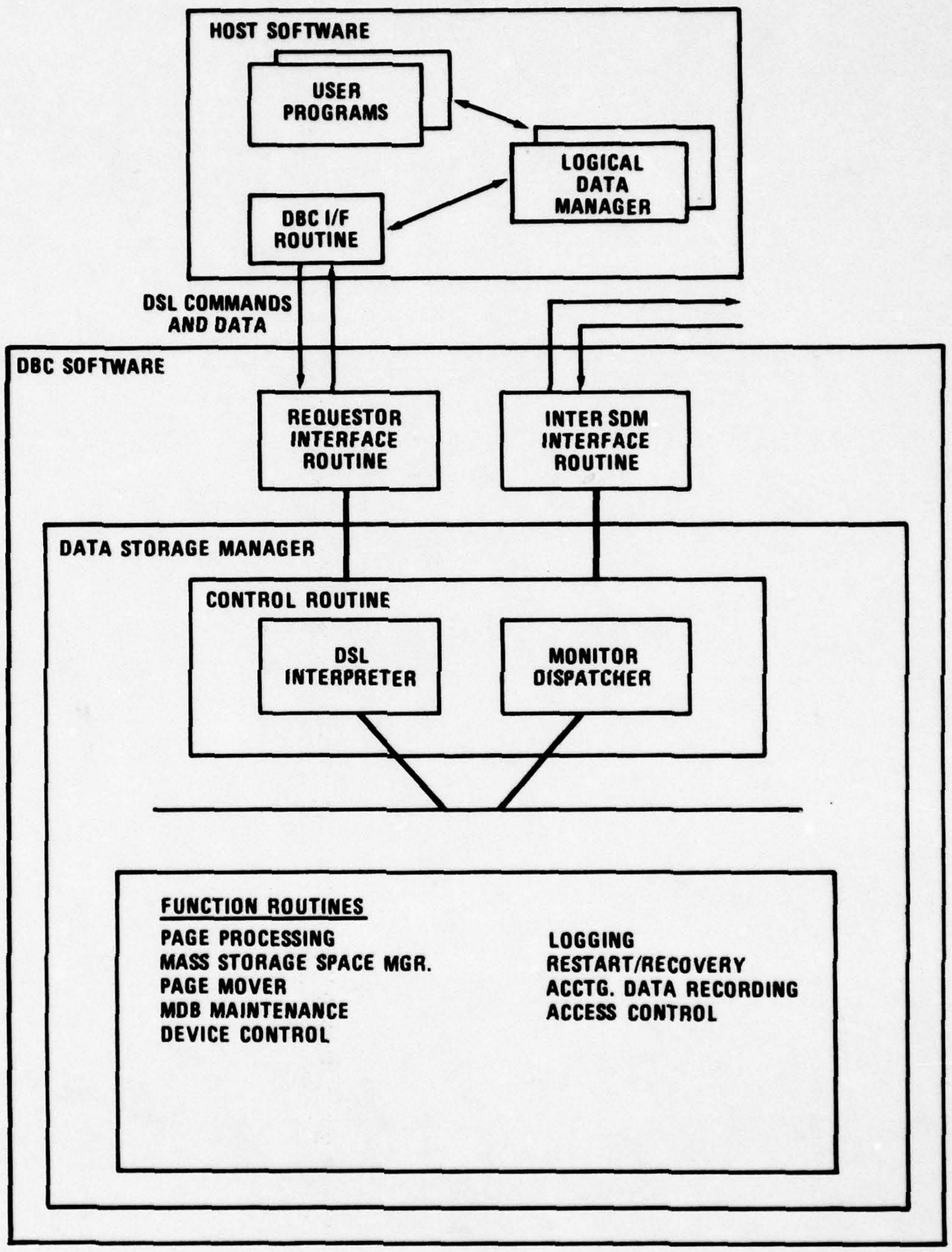


Figure 4-3. SOFTWARE ORGANIZATION

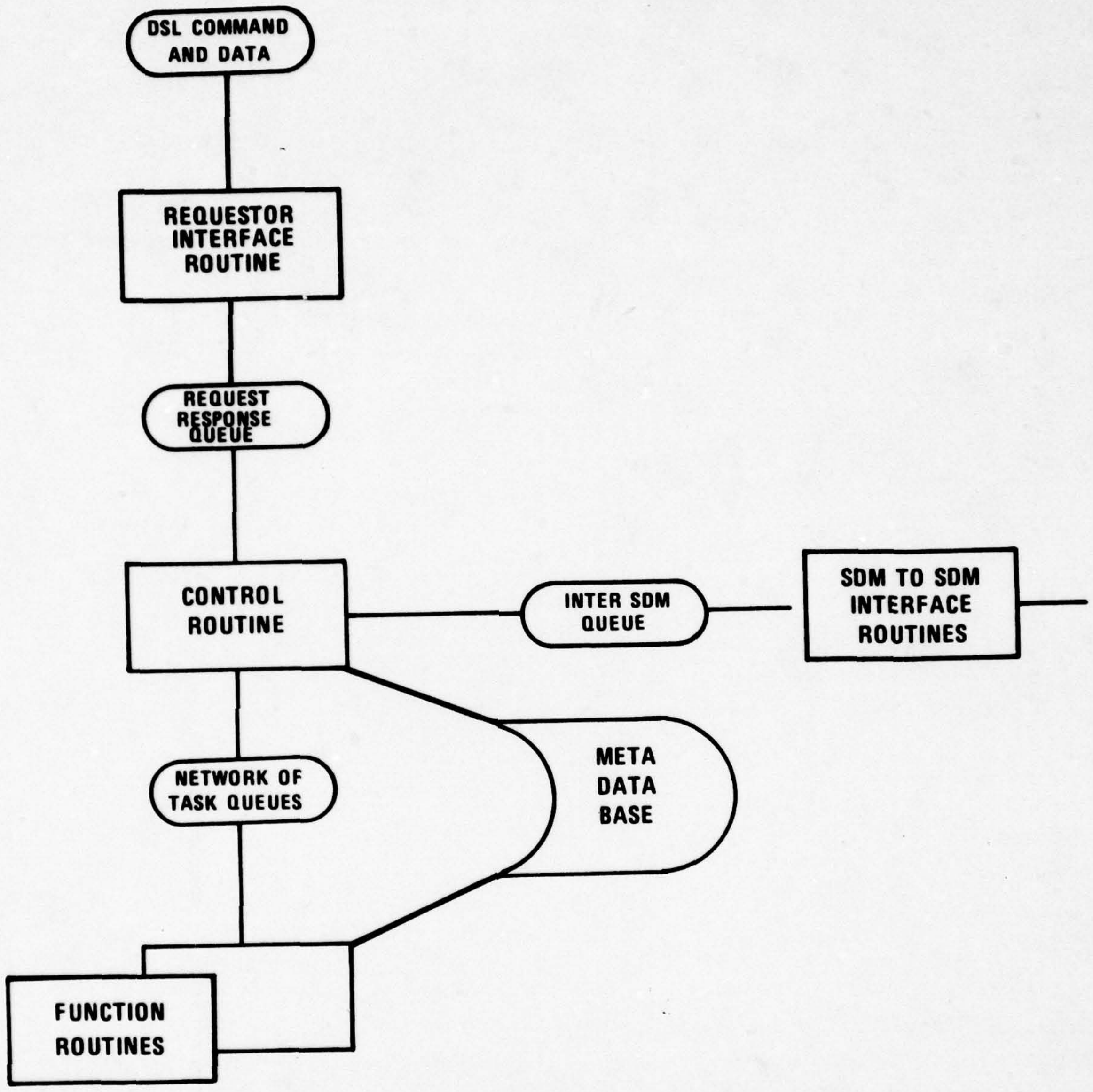


Figure 4-4. SOFTWARE INTERCOMMUNICATIONS

4.3.1. Host Software

The software that executes in the host consists of the user programs and Logical Data Managers (LDMs). There is a DBC Interface Routine for each host that handles the lower level protocols for intercommunication between the host and DBC.

4.3.2. Data Base Computer Software

The Storage Data Manager comprises the bulk of the DBC software. Specific interface routines also exist that handle the lower level protocols for intercommunication with the host and other Storage Data Managers operating in other DBCs.

The Storage Data Manager provides the logic to manage the storage areas and the data stored within them. There is a Storage Data Model that provides a common (or fundamental) data structure that all Logical Data Managers can use to store and retrieve data. The common data structure is not a super-set of all the logical data structures. It is, rather, a data structure which provides adequate fundamental storage structures with which an LDM can support its logical data model.

The Storage Data Manager consists of the following components:

1. Control Routine:

The control routine interprets the Data Storage Language commands utilizing the Meta Data Base and current control and locking information. The interpretation involves analyzing the command, expanding it as necessary, and setting up a network of primitive tasks to be performed. The tasks are queued to the function routines that must execute them.

The scheduler routine dispatches the tasks to the control and track processors and checks progress in executing tasks. Particularly it checks for the networked interactions between parallel executed tasks and the completion of the network of tasks representing a single DSL request. It then formats a response to the requestor.

2. Function Routines:

The function routines perform specific services. Tasks to be performed are obtained from the monitor routines in each processor which examines the task queues. Each function routine performs its function, moves data as appropriate and updates the task entry to reflect the status of the executed task.

Function routines are intended to be the lowest level subsystem entity. In general they do not call other function routines to perform services for them but rather return a status that the scheduler routing acts upon.

All the required function routines may not be identified yet. The following list represents the current design state:

- Page processing (search, update, order, insert, project, join, etc.),
- Mass Storage Space Manager,
- Block Storage Space Manager,

- Page Mover,
- Meta Data Base Maintenance,
- Device Control,
- Logging,
- Restart,
- Recovery of Data,
- Accounting Data Recording,
- Access Control and Security.

4.3.3. Interfaces between Major Software Subsystems

There are five interfaces between the major software systems at this stage of the design. Each represents a set of protocols for intercommunication of request, data, and response. Every request has a response. Each interface is via queues.

4.3.3.1. DSL commands and data

The Data Storage Language commands and data interface operate between the DBC Interface Routine in the host and the Requestor Interface Routine of the DBC. There may be multiple Requestor Interface routines in the DBC if necessary to support heterogeneous hosts. Each will accept its host commands, modify them as necessary and queue them.

4.3.3.2. Request/response queue

The Request/Response Queue is a queue of request packets and a buffer of data associated with the requests. The Requestor Interface Routine usually places DSL commands and data in the queue and extracts responses and data while the control routine usually extracts DSL commands and inserts responses. The control routine may insert unsolicited responses in the queue to communicate to the Logical Data Managers which are the requestors.

4.3.3.3. Inter SDM queues

When there are multiple DBCs in a configuration or network and when User Programs are permitted to distribute their data amongst these multiple data base nodes, the SDM software subsystems must communicate with each other. The functions of locking, deadlock detection or prevention, restart, and recovery require distributed data management control. The inter SDM queues exist in each SDM to hold the inter SDM requests and co-ordinate completion.

4.3.3.4. Network of task queues

This interface consists of a number of task queues, at least one per processor and possibly subqueues for function routines.

The complex is termed a "network of task queues" because each queue is not independent of the others. A particular DSL request is transformed into many tasks which are placed in the many processor queues. However the processing of tasks on any particular queue cannot proceed independently. There may be a sequence required such that a task in queue A cannot commence until a task in queue F is successfully completed. In another case N track processors may not be able to process a parallel operation until a set of tasks comes up to the top of each queue. The queues consist of task packets with space for response status and references to data as required.

4.3.3.5. Meta data base

The Meta Data Base is a set of files and records which contain the definitions and current descriptive information as described previously.

Elements of the MDB are created and maintained directly by a group of MDB maintenance function routines. These routines are driven by the Control Routine of the SDM and the DBA user level routines from the host.

The MDB is essentially a read-only table of definitions and directories as seen by the SDM routines. However, there is also a dynamic maintenance of the MDB occurring continously. Thus, routines using the MDB must be carefully isolated into "reference" and "maintenance".

4.4. Meta Data Base

4.4.1. Multi Level Data Definition

The SDM lends support to the notion of multiple levels of data definition:

1. the lowest level, the *internal* definition, describes how the data is represented and organized on the mass storage devices; it is specified in the *storage schema*;

2. the central level, the *conceptual* definition, provides the semi-permanent canonical description of the data, with respect to which all other definitions are directly or indirectly related; it is specified in the *logical schema*;

3. the higher levels, the *external* definitions, describe how the data is portrayed using a particular data model in a particular programming environment; it is specified in a *subschema*.

The SDM requires that there be: a logical schema describing the conceptual form of the data in terms of a chosen data model; and a storage schema describing the physical representation and organization of the data, and its mapping from the conceptual form. There may further be subschemas to describe the data in terms of the other data models (possibly for several different programming environments), and their mappings from the conceptual form. (Note that for most language files the storage, logical schemas and subschemas will be little more than a record description and may be simply generated.) The LDM and not the SDM will be concerned with the mapping between logical and external data. In many cases the portrayals will reduce to identity mappings straight from storage data through logical data to external data, so that the LDM need not intercept the data.

4.4.2. Data Dictionary

The heart of the data dictionary is an inventory of basic data objects which are the molds from which the building blocks are made for the construction of records, sets, files, relations and data bases.

4.4.2.1. Domains

A domain is a named abstract data type with a specific set of possible values. A domain has a very definite meaning which is identified through all levels of data definition. Thus, for example, Customer # and Supplier # are two distinct domains even though they may have overlapping or coincident ranges of representation values. Although domain is an abstract concept, each domain is constrained to have a single storage representation.

Each domain must have a definition of its storage representation, e.g. PIC H9(6). (The COBOL PICTURE clause may be a subset of the possible domain storage definitions.) The logical definition is normally identical to the storage representation, since item data transformation is not supported between storage and logical views. The only exception is that standard transformations will be allowed to convert values between incompatible systems, to convert: between one machine's floating point and to anothers floating point; between 8 bit ASCII and 9 bit ASCII; between 8 bit ASCII and 8 bit EBCDIC; and so on.

The external definition may also be identical to the storage representation provided that the programming language data types match it (as is likely for COBOL). A domain may have a different external definition, requiring that the LDM perform a transformation between logical and external representations. For instance PIC H9(6) might have a Fortran representation of INTEGER. Whether the external representation may be defined arbitrarily, or whether the program must accept the default representation for its programming language, is an open question.

Domains may be defined in terms of other domains. For instance, a group domain date may be built up from domains month, day and year, and primary-color could be defined as a subset of color.

4.4.2.2. Items

An item is a named data object whose possible values are taken from a specific domain identified by name. An item has very definite use which is identified through all levels of data definition. Thus birth-date has the same meaning whether it appears in a patient record or in a hospital employee record or both.

A group item is a named group of elementary items whose use is normally determined as a group. Alternatively it may be defined like an elementary item, but from a domain built up from other domains.

4.4.2.3. Validation

Validation rules may be specified either for items or for domains to apply to all items defined for that domain. Different validation policies may be applied, for instance: validate every new value; or validate once a week.

4.4.2.4. Cross references

Cross referencing information is maintained in the data dictionary showing:

- which items use particular domains;
- which records use particular items;
- which schemas use particular records;
- which subschemas use particular schemas;
- which programs use particular subschemas.

4.4.3. Directory

The Directory is a catalogue of storage area occurrences. The multiple occurrences might be successive cycles of a file or different occurrences of a data base structure for different users. The Directory is where the SDM maintains: the current locations of storage areas; usage statistics; etc..

4.4.4. Logical Schema

A logical schema is the description of a conceptual data base.

- For a relational schema it defines the form of:
  - the relations (logical record types) in terms of the items available in the inventory of the data dictionary;
- for a network schema as per the proposed CODASYL standard, it defines:
  - the logical record types, as above,
  - the (logical) sets, and
  - the (logical) areas.

In this software section, several different views of a single data base are illustrated. The notation used is as follows:

- upper case letters indicate record types:
  - an untagged letter indicates a logical relation or record type, e.g. A;
  - a primed upper case letter indicates a storage record type, e.g. A', B''.
  - an asterisked upper case word indicates an external (sub-schema) record type, e.g. A*, BD*;
- an upper case letter followed by a digit indicates a key:

- the digit 1 indicates a primary key, e.g. A1;
- the digit 2 indicates part of a secondary key, e.g. D2;

- an upper case letter followed by # indicates an encoded data item, e.g. A#;
- a lower case letter, possibly followed by a digit, indicates multiple data items not contributing to keys, e.g. b1, d;
- a hyphenated uppercase word indicates an index or set; thus C-D' is the secondary item index supporting the set C-D, and A#-A1' is the value encoding index supporting the encoding of key A1 into A#.

4.4.4.1. Example network logical schema

Figure 4-5 illustrates a Network Logical Schema which has: four logical record types A, B, C, D; and three sets A-B, B-D, C-D. As it happens this definition has no apparent redundancies, that is, no data item is duplicated across records.

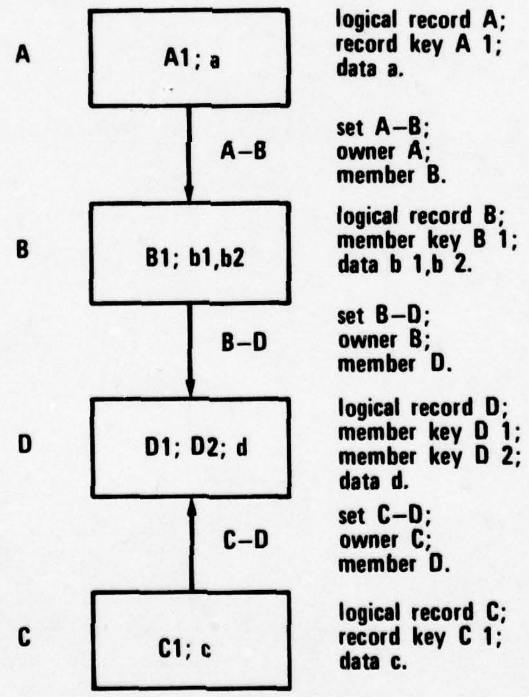


Figure 4-5. NETWORK LOGICAL SCHEMAS

4.4.5. Storage Schema

The storage schema is the definition of the stored form of the data base and its mapping to the logical schema.

- It describes the forms of:
  - the storage records in terms of the logical records in the logical schema,
  - the storage areas to hold the storage records,
  - and the secondary indices to provide rapid location of the storage records;
- it defines the mappings between:
  - storage records and logical records,
  - secondary indices and sets,
  - storage areas and logical areas.

4.4.5.1. Storage records

The characteristics of storage records are specified by the DBA in the storage schema:

- the name of the storage record;
- the name of the logical record from which the storage record 's projected;
- the names of the component items constituting the storage record, in sequence;
- the names of the items making up the primary key (including a key suffix if necessary) and for each item whether the item is ordered in ascending or descending sequence;
- the names of the secondary keys and their constituent items;
- the definitions of any system suffices required to provide uniqueness of primary keys or to order within secondary keys.

4.4.5.2. Storage areas

The characteristics of storage areas are specified by the DBA in the storage schema:

- the name of the storage area;
- the name of the logical area which covers the storage area;
- the initial and maximum number of pages reserved for the storage area;
- the initial load factor for the pages;

- the rules for allocating pages as the storage area becomes populated with records;
- the provisions to be made to enable recovery (rollback and/or reconstruction) of the storage area;
- which record type or types are contained in the storage area; from this may be deduced:
  - whether the area contains any records of varying length;
  - for storage areas containing only fixed length records, whether they are all of the same length.
- whether the records are ordered (by primary key) within each page;

4.4.5.3. Indices

The characteristics of the secondary indices are specified by the DBA in the storage schema:

- the name of the index;
- the type of the index, item or range;
- the storage record types supported by the index;
- the (logical) set supported by the index, if applicable.

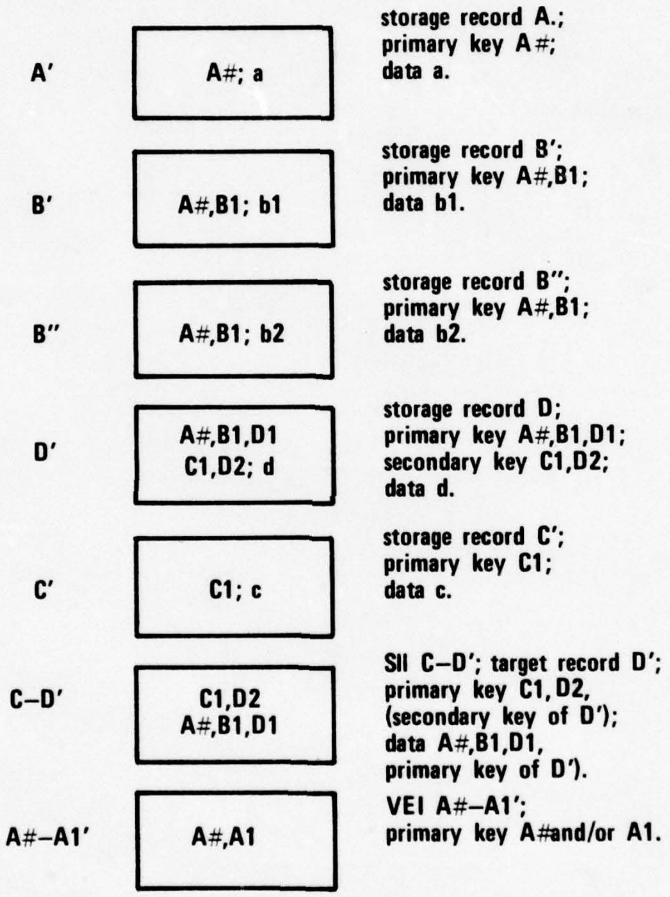
The characteristics of the value encoding indices are also specified in the storage schema:

- the name of the index;
- the size of the system code.

The characteristics of the primary range indices are wholly determined by the definition of the storage record.

4.4.5.4. Example storage schema

Figure 4-6 illustrates a Storage Schema which has: five storage record types, A', B', C', D': one secondary item index, C-D'; and a value encoding index A#-A1 . Logical records A and C are represented by the storage records A' and C'. The record keys of A and C, A1 and C1, are adopted as the primary keys of A' and C'. The logical record B has been split into two storage records B' and B'' and its membership in set A-B represented by incorporating the owner's A's record key, A1. The primary key of both B' and B'' is then the combination A1, B1. The logical record D is represented by the storage record D', and its membership in sets B-D and C-D represented by incorporating the primary keys of the owners. The combinations A1, B1, D1 and C1, D2 become the primary and secondary keys respectively. The sets A-B and B-D, being primary paths, are supported by the PRIs for B' and D'. The set C-D, being a secondary path, is supported by the SII, C-D'. The key A1 has been encoded into A# via the VEI, A#-A1'.



storage record A.;
primary key A#;
data a.

storage record B′;
primary key A#,B1;
data b1.

storage record B″;
primary key A#,B1;
data b2.

storage record D;
primary key A#,B1,D1;
secondary key C1,D2;
data d.

storage record C′;
primary key C1;
data c.

SII C–D′; target record D′;
primary key C1, D2,
(secondary key of D′);
data A#,B1,D1,
primary key of D′).

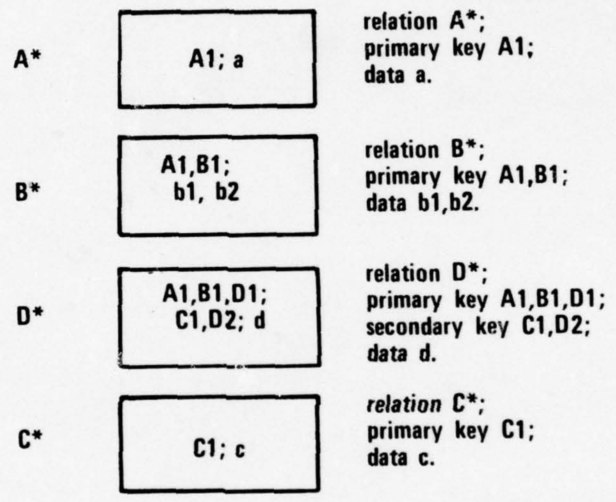
VEI A#–A1′;
primary key A#and/or A1.

Figure 4-6. STORAGE SCHEMA

4.4.6. Subschemas

4.4.6.1. Example relational subschema

Figure 4-7 illustrates a Relational Subschema which has four relations A*, B*, C*, D*. These map closely onto the storage records A', (B',B''), C', D'. In fact the Relational Subschema could have been the by-product of the Storage Schema compilation.



```
relation A*;
primary key A1;
data a.

relation B*;
primary key A1,B1;
data b1,b2.

relation D*;
primary key A1,B1,D1;
secondary key C1,D2;
data d.

relation C*;
primary key C1;
data c.
```

Figure 4-7. RELATIONAL SUB-SCHEMAS

4.4.6.2. Example network subschema

Figure 4-8 illustrates a subschema which provides an alternative network view of the logical schema. For this subschema, logical records B and D are viewed as one record type, BD*, and the sets A-B, B-D are collapsed into one, A-BD*.

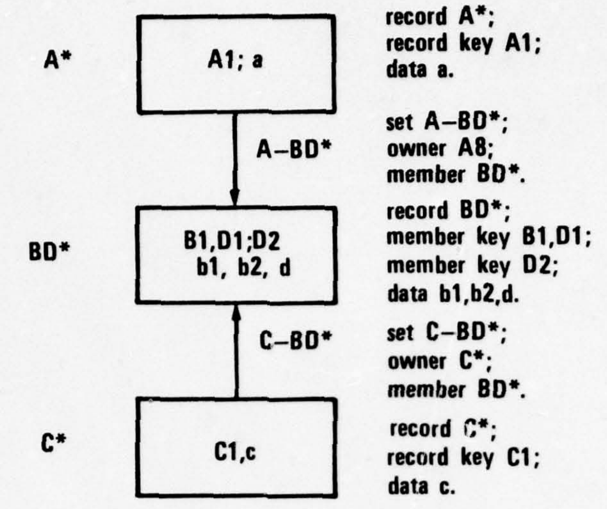


Figure 4-8. NETWORK SUB-SCHEMA

4.4.6.3. Example ISAM subschema

Figure 4–9 illustrates an ISAM Subschema for which all four logical records are collapsed into one record type, ABCD*. The set relationships are all represented in the single record type.

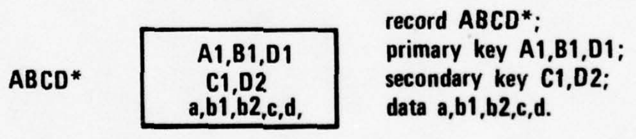


Figure 4-9. ISAM SUBSCHEMA

4.5. Storage Structures

It is the function of the Logical Data Managers to provide the Network, Relational, or whatever views of the Data Base, while the SDM provides sophisticated storage data services to allow this. The underlying data structures used, which are essentially content addressable, provide excellent support for other data models including conventional language files and access methods.

4.5.1. Storage Hierarchy

The storage managed by the SDM on the DBC may be regarded as a hierarchy:

1. data bases – a DBC may hold several separate data bases;

2. storage areas – each data base is contained in several distinct storage areas;

3. pages – each storage area is made up of several distinct pages;

4. tracks – each page is made up of a predetermined small number of distinct tracks;

5. sectors – each track is made up of a predetermined small number of sectors so as to reduce latency;

6. storage records – one or more (usually many) storage records may fit into a single sector;

7. items – each storage record consists of one or more items which may be elementary or group.

4.5.1.1. Storage areas

A *storage area* is a named subdivision of the mass storage space allocated to a date base. It is a container for storage records.

Storage areas provide a convenient partitioning of a date base which may reflect device or geographic distribution desirable for performance, recovery, or security reasons. There may be a many-to-many correspondence between storage areas and storage record types. A storage record type may be distributed across several storage areas. A storage are? may contain several record types. Commonly however, a single storage record type will be contained in a single storage area. Storage areas of a single record type permit smaller indices and faster arbitrary searches, but storage areas of mixed record types may provide advantages in the context of multi-member sets within the Network Data Model and for small files.

4.5.1.2. Pages

A *page* is an individually numbered subdivision of the mass storage space available to the DBC. The page is both the unit of allocation for storage areas and the unit of transfer between mass storage and the SDM buffers.

The size of a page is device dependent and is the product of the track size for the device and the number of tracks which may be transferred in parallel. It is expected that a page will be three or six tracks from a single cylinder.

4.5.1.3. Tracks and sectors

The track, of course, is that part of the mass storage available through one head at one position. It is expected that the size of the track will be 50 kilobytes.

A track will be divided into a small number, perhaps 8, of equal sized sectors. This will reduce the data density only marginally and will reduce the rotational latency significantly.

4.5.1.4. Storage records

A *storage record* is a variable length content-addressable container for data. A storage record may be a projection or a subset of a single logical record. The contents of a storage record are defined in the storage schema.

A fixed length record type may be regarded as a two dimensional table with the rows as the records and the columns as the elementary items. The table may, of course, be arbitrarily large. The rows of the table may be conceptually in arbitrary order. The table is then very like the relation concept of the Relational Data Model.

Any record type may also be regarded as a file of records. The DBA may specify that the record type be ordered by primary key. The ordering may be of no interest to the Relational LDM but should improve response to the SAM LDM.

Note that on the DBC, a storage record is wholly contained within a single sector. However a logical record may be defined as being made up of several storage records to any arbitrary size.

4.5.1.5. DBKs

Network Model Implementations support the notion of *Data Base Key (DBK).* There is no universally accepted definition of DBK, but it is usually taken to mean a unique integer identifying a particular record occurrence, the use of which implies faster access than other types of record selection.

On the DBC, records do not contain any inter-record pointers or Data Base Keys, either visible to or hidden from the user. In fact the concept of Data Base Key does not even exist on the DBC. (A view including Data Base Keys may, of course, be provided for existing programs which depend on them, but the performance of Direct Access is now achieved without then)

4.5.1.6. Primary keys

The SDM relies on unique *Primary Keys* to identify records. A unique Primary Key (PK) consists of one or more items in the record (or columns of the table). A PK must be unique so that every record may be unambiguously identified. However not all items in the PK need be visible to the user so that he may view the record as allowing duplicate Primary Keys. A storage record type which is distributed over several storage areas or DBCs must have included, as the most significant part of its PK, the parameter which determines the storage area. For instance, for an employee record type, distributed across storage areas, the PK might be plant #, employee #.

A record is addressed by searching for the one containing the appropriate PK. Hence the term "content addressing". The search will be limited to a single page. For unordered records the search will be conducted in parallel using the track processors and the key processor. For ordered records the search will be conducted by a single track processor once the other track processors have determined that their subranges are inapplicable. Of course, in searching for collections of records satisfying some arbitrary predicate all track processors will normally be involved. Note that the PK of a record, once stored, may never be changed. If a Data Model supports modification of Primary Keys, the corresponding LDM must issue a special MODIFY-PK command which the SDM will effect as a DELETE followed by a STORE.

4.5.1.7. Advantages

The advantages of a Data Structure using embedded PKs instead of DBKs, are as follows.

- The resulting *tabular form of the records is easy to visualize and understand* – this is illustrated graphically by comparing Figure 4-10 a Network Data Base of 12 record occurrences, with Figure 4-11 the equivalent Content Addressable Data Base but with 27 record occurrences; the fact that all SDM system tables have the same simple form, will increase utility of the SDM primitive functions; this will contribute to system compactness and robustness.
- It is amenable to arbitrary retrieval predicates.
- It facilitates the support of dynamic views; a set or a record may be easily fabricated for the duration of a run.
- It facilitates physical reorganization even on the fly, since a record may now be completely relocated without having to "fix up" pointers to it from other records.
- It facilitates rollback and reconstruction.
- It facilitates software fault detection, since a record's set participation can be determined from its content, without any need to follow possibly contaminated chains.
- !t is amenable to hardware acceleration.

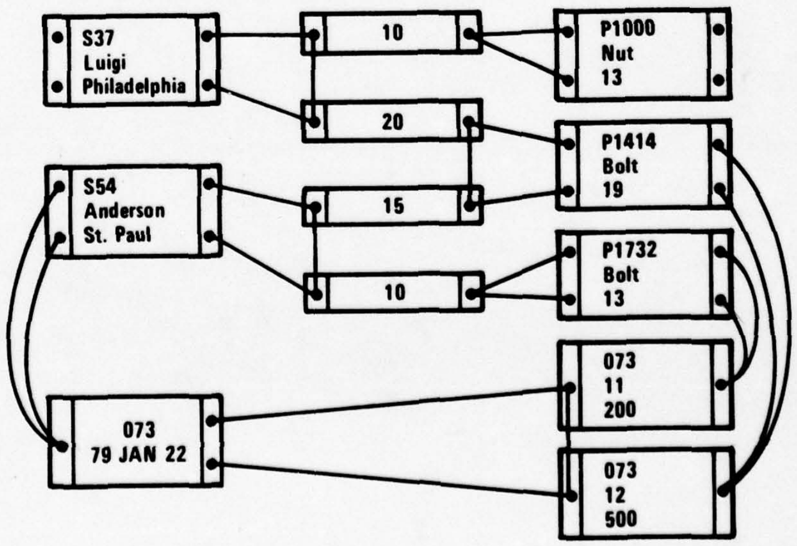


Figure 4-10. NETWORK DATA BASE - OCCURRENCE DIAGRAM

Supplier

S #	S Name	City
S37	Luigi	Philadelphia
S33	O Leary	Sudbury
S41	Smith	Salt Lake
S54	Anderson	St. Paul

Supplier-Part

P #	S #	Price
P1000	S37	10
P2449	S33	12
P1414	S37	20
P2449	S37	15
P2236	S37	10
P1414	S54	15
P1732	S54	10
P1732	S41	10
P1000	S41	9
P1000	S54	7

Part

P #	P Name	Size
P1000	Nut	13
P1414	Bolt	19
P1732	Bolt	13
P2449	Nut	19
P2236	Washer	19
P2000	Washer	25

Order

S #	O #	Data
S54	062	78 Dec 28
S33	067	78 Dec 29
S54	073	79 Jan 22

Item

S #	O #	1 #	P #	Qty
S54	062	11	P1000	200
S54	073	11	P1732	200
S33	067	11	P2449	500
S54	073	12	P1414	500

Figure 4-11. CONTENT ADDRESSABLE DATA BASE – OCCURRENCE DIAGRAM

4.5.1.8. Disadvantages

The disadvantages are as follows.

- Existing Data Bases will require reorganization.
- Some Data Bases will occupy more space; however initial studies indicate that this may only be a few per cent; moreover other Data Bases will actually occupy less space.
- LDMs will have to perform some command translation in order to support existing programs.
- New hardware and software techniques must be developed for the content addressing to be effective.

4.6. Logical Data Models

4.6.1. Access Methods Support

Different Access Methods may be supported by the Primary Key technique. An appropriate Primary Key is identified for each, and some software implications examined. In general, it is presumed that existing data will require processing by a Reorganization Utility before becoming available on the DBC.

Whenever record types are mixed within an area it will be necessary to include the Record Type # (Record Code) as part of the Primary Key behind the scenes. But in any case there are other reasons for embedding the Record Code in the record.

4.6.1.1. SAM

For Sequential Files, the most natural Primary Key is the ordinal number of the record on the file. This Primary Key is system generated and managed, but the limited repertoire of SAM functions makes this straightforward. Since all access of the file is sequential, the DBA would specify the storage record to be ordered.

4.6.1 2. DAM

For Direct Files, the natural Primary Key is the *ACTUAL KEY* (COBOL) or *DBK* (DMS). This would be embedded in the storage record and exposed to the user as required. This provides the requisite random access. However the key would not relate directly to any physical address. It also allows sequential access for the FIND NEXT WITHIN AREA of the network model. If sequential access of the file is frequent, the DBA would specify the storage record to be ordered.

4.6.1.3. ISAM

For Indexed Sequential Files, the natural Primary Key is the ACTUAL KEY (COBOL) or the *Record Key* (DMS). This is usually already embedded in the user's view of the record. If, for the Record Key, duplicates are allowed, a Key Suffix must be appended to the Record Key to complete the PK. Since the user has no control over the ordering of records with duplicate keys, the automatic management of the Key Suffix is straightforward. If sequential access of the file is frequent, the DBA would specify the storage record to be ordered.

4.6.1.4. CALC

For CALC records, the natural Primary Key is the Record Key. The SDM may treat CALC records in the same way as ISAM, except that there is no requirement to order CALC storage records.

4.6.2. Network Data Model Support

The Network Data Model is most important as it is being defined and developed by CODASYL to be the industry standard data model.

4.6.2.1. Key propagation

In representing a Network Data Model, for which implementations have traditionally relied on inter-record pointers, on a content addressable storage structure, alternative means must be provided to represent set relationships.

The process is conducted in essentially two stages.

1. Identification of all root record types – the root records are those record types for which primary keys can immediately be determined.

2. Propagation of owner keys – the primary keys of root records are propagated through all sets that they own to be embedded in the member records; the combination of the owner key and the member key becomes a secondary key, or the primary key if the member is located via that set; the newly formed primary keys are then further propagated down through any set which the members own; and so on.

Non-existence of member keys or the allowance of duplicate keys complicates the issue but the resultant problems are resolvable. Figure 4-12 shows a simple Network Schema suitable for an Assembly Plant to control its parts' supply. Parts have numbers, names and sizes. Suppliers have numbers, names and cities. Each part may be supplied by several suppliers at particular prices. Each supplier supplies several parts. Orders are placed with suppliers on particular dates. Orders consist of several items, each for a quantity of parts.

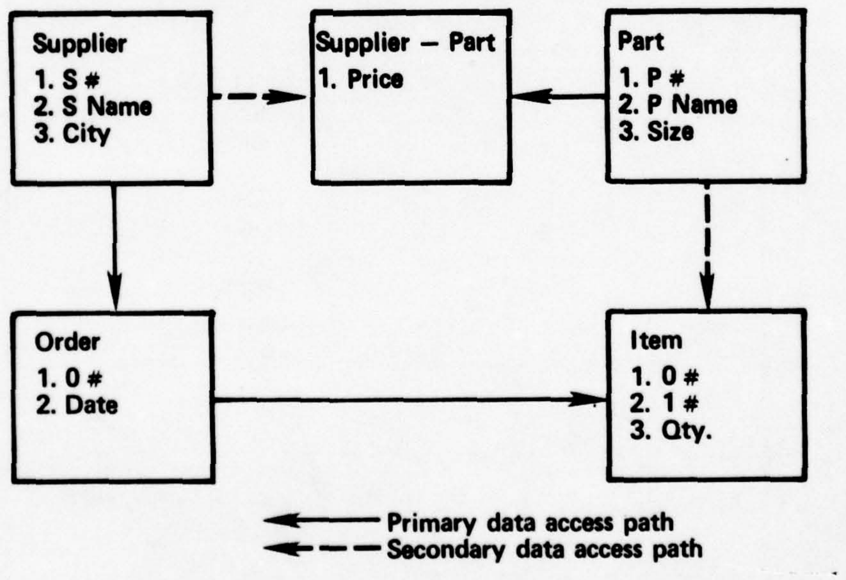
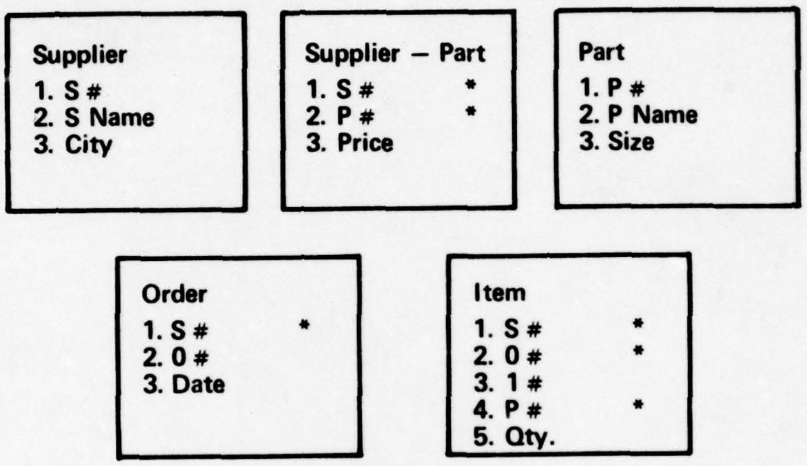


Figure 4-12. NETWORK SCHEMA

Figure 4-13 shows the same schema in a content addressable form. The propagated keys are indicated by an asterisk.



– Keys are propagated to records lower in the network structure (*)

– No pointers

Figure 4-13. CONTENT ADDRESSABLE SCHEMA

4.6.2.2. VIA set (primary path)

For records located via set, the Primary Key consists of two parts:

1. the first part identifying the set occurrence of which the record is a member;

2. the second part identifying the record within the set occurrence.

The first part consists of a key propagated from the owner record of the set. The propagated key will be the owner's Primary Key for set occurrence selection through location mode of owner or through current of set, but one of the owner's Secondary Keys for other path selections.

The second part of the member's Primary Key is the Set Key for a sorted set (extended by a Key Suffix if duplicates are allowed) or a Key Suffix alone for an unsorted set. Here the management of the Key Suffix must be more sophisticated because it is not always possible to predict the insertion sequence. For set insertion order FIRST or LAST, successively generated Key Suffices may form a descending or ascending sequence and there is no problem. For set insertion order NEXT or PRIOR, any new Key Suffix must lie in the range of its neighbors and in general the algorithm must react to the insertion pattern. The problem is greatly alleviated if many records are stored at a time. Bulk loading utilities will be provided to exploit this. Very occasionally Key Suffices within a single set occurrence will have to be reorganised to allow breathing space in cramped ranges.

If sequential access of the set is frequent, the DBA would specify the storage record to be ordered.

4.6.2.3. Secondary paths

Secondary paths to records are supported by Secondary Item Indices, see earlier section. Each entry of the SII will be made up of the secondary key and the primary key both of the member record. The secondary key will be formed in very much the same way as the primary key was formed for the primary path.

Network set operations are supported by essentially the same techniques used by DMS-1100 with Pointer Array Sets. Inserting a record is performed by storing a new SII record; finding next within set is performed by first finding next within SII record. Key Suffices will have to be included in SII records for unsorted sets and for sorted sets which allow duplicate keys, in order to ensure uniqueness of SII primary keys. The key suffix and the physical ordering will represent any incidental ordering established by user programs. Exactly what constitutes the secondary key for the member record is determined by its own characteristics together with those of the set and its owner.

The DBA may choose not to embed the secondary key in the target storage record, but on retrieval to perform a join across the primary key of the target storage record and its SII to form the target logical record.

4.7. Hierarchic Data Model Support

The Hierarchic Data Model is important mainly because it is the application programmer's view of the Information Management System (IMS) currently marketed by IBM.

The Hierarchic Data Model is, in fact, a subset of the Network Data Model.

- a segment maps to a record type;
- a field maps to an item;
- a child segment type maps to a set member;
- a parent segment type maps to the set owner.

(The Network Data Model representation of a Hierarchic Data Model would have no set with more than one record type as member, and no record type as a member of more than one set.)

In fact, the Hierarchic Data Model is even closer to the content addressable representation of a Network Data base.

- the concatenated keys are the primary keys formed by propagation of owner keys in the propagation process described above;

- a hierarchic model record type maps into the record type defined by propagating the complete owner record into each of its members, and so on down the hierarchy, as was previously in the example ISAM Subschema.

4.7.1. Support of Total's Data Model

TOTAL is one of the most widely used Data Base Management Systems. TOTAL uses a form of network model limited by the restriction that members of sets *(linkage paths)* may not themselves own sets.

The TOTAL's Data Model fits very easily into the Data Base Computer Storage Structures. A *master (single entry) data set* maps on to an owner record type with the *control key* becoming the primary key. A *variable entry data set* maps on to a member record type. The control keys of master data sets are by definition already embedded in the variable entry data sets owned by them. A primary set must be chosen from the sets defined. A key suffix must be included in the member record type to complete the primary key. Key suffices must be included to reflect the ordering in secondary sets.

4.7.2. External Views

The following outlines some of the possible mappings available which the LDM may exploit in portraying its view of the data. In general the view may be defined semi-permanently in a data model dependent logical schema or defined temporarily for the duration of the run. The Network LDM may portray the data base in terms of records and sets, the Relational LDM in terms of Relations, and so on.

This section also considers the mappings which the DBA may exploit in optimizing the storage structure of the data base, whilst maintaining a constant logical structure.

It will be seen that these mappings fit elegantly into the structure of the Data Storage Language, see later section.

4.7.2.1. Projection

An external record may be defined in a sub-schema as a subset (or a projection) of a logical record, thus limiting the items visible in the external record. The projection may be with or without duplicates elimination, according to the data model requirements. Key suffices would always be eliminated from user visibility. Note that primary keys may not be projected out if updating is to be allowed and that rules must be provided in the logical schema for initialization of logical items (and hence of storage items) which are not visible in external records.

4.7.2.2. Selection

An external record may be defined as a selection of a logical record. An end user facility could define an external record as the selection of logical records satisfying some criterion, e.g. all employee records with grade eleven.

4.7.2.3. Join

An external record may be defined as the join of two logical records across a common domain. The external record might commonly be determined by the requirements of a particular report.

4.7.3. Special Join

There is one special case of the join operation, which is referred to here as *marriage*, that deserves special attention for two reasons:

1. it is particularly easy to perform and is common enough to warrant optimization;

2. relations formed by marriage, unlike those formed by arbitrary joins, may be updated without anomaly.

The marriage is the join of two relations across their common primary key. Each record occurrence in one operand has no more than one (and sometimes exactly one) matching partner in the other operand.

The logical record might be the complete information about an employee formed by marrying two storage records which happen to be autonomous due to their different frequencies of use or different security or recovery requirements.

4.7.3.1. Others

Mapping may further be defined as combinations of the above relational operations, projection, selection, and join, together with the set operations of *union, intersection, and difference*.

4.8. Recovery

Recovery is a broad term that deals with the ability of the system to overcome error conditions. The key functionality of the database computer is the ability to maintain consistency and to reconstruct a data base to the state it had at any instant of time.

The global function of recovery will be broken down into categories or sub-functions in the DBC.

- In-line recovery from errors
- Recovery of directories and indices
- Consistency and rollback
- Reconstruction of part of a data base
- Recovery of a data base to a specific state

The Storage Data Manager (SDM) will perform some of these functions automatically based on storage schema definition.

4.8.1. Error Detection

Naturally there will be extensive logic to detect error conditions. Each subsystem should have tests of its input requests and its local data objects, which can be activated or not. Each should also have a minimum level of error checking. Where possible in-line corrective measures should be applied. When in-line correction is not feasible, recovery procedures should be automatically scheduled and corrupted data or logic locked out.

Statistics on error recovery attempts and successes should be an inherent part of the in-line performance monitoring feature with appropriate threshold reporting and alternate error recovery strategies.

4.8.2. In-Line Error Recovery

There is a class of error conditions that may be handled in-line without serious system degradation. These are recoverable hardware errors and self-correcting software algorithms. The error recovery routines should generally be non-resident and loaded when needed.

4.8.3. Recovery of Directories and Indices

Directories define the state of a data object, point to its physical storage, and maintain activity history (accounting, error statistics, utilization). Thus Directories aren't reconstructable from the data itself. However, Directories must be reconstructable. This implies a combination of duplexing the state and physical information and logging activity history.

Indices are abstractions of the data and must contain no information that can not be reconstructed from the data base itself. Thus indices need not be maintained reconstructable, except, duplexing could be employed by a DBA to reduce recovery time. Recovery of indices should be scheduled when errors are detected.

4.8.4. Consistency and Rollback

The processing of a transaction (or a batch task) may perform certain update operations on a data base. The purpose of the transaction is to cause a change in the state of the data base from its state immediately before the transaction to an updated state.

For purposes of maintaining consistency of the data base state throughout the processing of a transaction (or batch task) and in order to provide rollback of the state, consistency and locking procedures should be employed.

Multiple transactions (or batch tasks) can be permitted to change the state of a data base concurrently providing the data records and relationships being updated are disjoint. The technique to accomplish this disjoint updating is the technique of locking the data objects and relationships required to assure consistency for a transaction.

Thus it can be said that updating the data base is a concurrent stream of discrete, disjoint changes of state.

Because of the need to vary the scope and time span of the locking tasks, they are defined into manageable units, as follows:

■ Transaction -	a single task performed by a host which may result in many data storage accesses and updates.
■ Batch Task -	a single task which processes a batch of transactions For consistency purposes it is treated like a long transaction.
■ Conversation -	a string of transactions.
■ Consistency Unit -	a unit of work which spans a variable period of time during which the portion of the data base which it references for update, or updates, is locked for the exclusive use of this consistency unit.

- A consistency unit can be:
  - a transaction
  - a conversation of many sequential transactions
  - a batch task
  - a part of a transaction or batch task, when the transaction logic issues explicit commit commands breaking itself into a series of consistency units.

The scope of the data locked will vary from the records currently in use or previously updated, to all the records in the data access paths referenced. The trade-off between locking too much and creating bottlenecks or not locking enough and encouraging deadlocks needs further study. Explicit lock commands should be available to lock predicates for complex operations or entire record types (relations).

Updates should not be performed directly against the data base. Instead, records and relationships should be updated in the abstract set in block storage (or extended block storage on disk). When a consistency unit issues an explicit or implicit commit, the abstract set is marked as committed and it is scheduled for data base update. The locks are not released for future update until the data base update has been successfully accomplished.

4.8.5. Reconstruction

Reliability and availability of the data bases are the highest priority of the SDM software. One of the mandatory requirements is that any part or all of a data base must be reconstructable. This primary principle requires that there is always a current copy of each part of each data base - and a redundant source or sources of information which the reconstruction routine can use to rebuild it. Some of the redundant sources which can be utilized are:

- ■ duplex current copy
- ■ dynamic checkpoint and after look log
- ■ static checkpoint, transaction logs (or batch transaction file), and the applications programs.
- ■ series of data base states and state changes consisting of checkpoints, before look log, and after look log.

Both checkpoints and logs are based on primary keys (instead of physical addressing) which permits dynamic reorganization.

The SDM is responsible for creating the redundant source data objects (except the transaction log) as an integrated part of its administration and update processing. The redundant source data possibilities are:

- Duplexed copy

  A copy of the data base maintained on a separate device. Updates are simultaneously applied to both. Either may be used for retrieval.

- Static checkpoint

  A copy of the data base part will be made, most likely to a tape or cartridge. For a static checkpoint the part is locked during the copy process.

- Dynamic checkpoint

  A copy of the data base part is made, most likely to a tape or cartridge. For a dynamic checkpoint only a page of the data part will be locked as it is copied. Each page is time stamped as it is copied and the time stamp is used in conjunction with before and after looks for reconstruction.

- After looks

  The version of a record after an update commit is logged. In conjunction with a checkpoint a current copy of the part can be created.

- Before looks

  The version of a record before an update is logged. In conjunction with the current state or some previous states, the before looks can be utilized in reverse order of occurrence to roll back to a prior state.

- Logged transactions

  Where re-processing may be required, input transactions must be logged explicitly by the host.

  In the case of batch operations, those files designated as transaction files should be flagged for saving rather than copying them into the log.

4.8.5.1. Reconstruction of part of a data base

This reconstruction function is oriented towards reconstruction of a part (or, of course, all) of a data base to its latest state. It is assumed that this part of the data base was found to be in error and the requirement is to reconstruct it to its latest state. It is also assumed that the bad part was locked out or removed from availability as soon as the error was detected.

Not all detected errors can be recovered with this approach, particularly if the error is of the type that is not immediately detected. More specifically, errors for which a check is made on each access to data meet this category. Examples are disk read errors, checksum errors, etc..

In order to reduce the time to reconstruct under this mode, frequent checkpoints should be taken. This reconstruction may then be effected by locating the latest dynamic checkpoint and performing a co-ordinated reconstruction from the time phased checkpoint records and after looks following them in time.

4.8.5.2. Recovery of a data base to a specific state

This function deals with reconstructing a data base of a particular state in time. It is only applicable to a complete data base since this is the only way to assure consistency at that state in time.

There are two approaches:

- Start with the current data base and go backwards in time applying before looks.
- Load a checkpoint prior to the desired state and apply the after looks forward in time.

A consistent state is defined as a point where no update activity was in process. Therefore in each case above, once a time state is reached, the consistency units that were active at that time are rolled back. The logs are then positioned just after the last after look and any open transactions are brought forward.

5. DATA BASE COMPUTER PERFORMANCE

Analytical calculations of the data base computer performance were made and are presented in this section on performance. The performance figures that follow represent the first cut at determining performance. The next step in our data base computer research is to build a simulation model and to construct a test vehicle. Performance estimates obtained from these efforts will offer a more accurate view of the data base computer performance.

5.1. Assumptions

1. Primary clustering in the DBC: All occurrences of the same record type are placed in as few cylinders as possible.

2. Secondary clustering in the DBC: This is based on the location mode of a record type. Even if all occurrences of a record type were to occupy $j > 1$ cylinders, if the location mode is direct, then, given a data base key, the specific occurrence can be tracked down to a single cylinder. If the location mode is calc or index sequential, then a specific record occurrence can be found in a single cylinder access as long as the calc key or a value of the indexed attribute is provided. If the location mode of a record type is via a set, then all member records of the same type and belonging to the same set occurrence are stored together in the same cylinder and are identified by their owner's data base key.

3. Only data base accesses (including index or structure memory access) are considered in the performance analysis.

4. Predicate checking in the host computer is assumed to consume no time.

5. Loading the Key Processor (KP) (of the DBC) with a set of values is assumed to consume no time.

5.2. Nomenclature

p	=	average physical block size used in current conventional mainframes
	=	448 bytes
c	=	amount of data accessed in one disk revolution by the DBC
	=	capacity of nine tracks of Ampex's Parallel Transfer Disk
	=	181,440 bytes
m	=	ratio of DBC's disk access and processing time to mainframe's block access time
	=	(1/2 revolution latency + 1 rev. data transfer + 1 rev. processing time)/(1/2 rev. latency + 1/2 rev. data transfer)
	=	2.5

s = ratio of DBC's structure memory access and processing time to mainframe's block access time

= 0.2

M = # of STUDENT records in the data base = 15000

avg. # of sections per course = 2

avg. # of courses taken by each student = 4

avg. # of courses taught by an instructor = 3

N = # of COURSE records in the data base = 1000

I = # of INSTRUCTOR records in the data base = 700

length of a STUDENT record = 250 bytes

length of a CURR-CLASS record = 15 bytes

length of a SECT-REC record = 30 bytes

length of a COURSE record = 60 bytes

length of an INSTRUCTOR record = 250 bytes

avg. # of students per section = $\frac{15000 \times 4}{1000 \times 2} = 30$

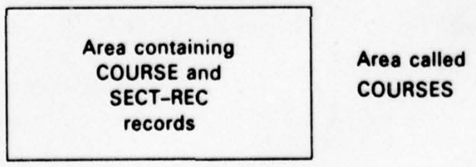
5.3. Record Retrieval

5.3.1. Record Occurrence Scanning

Category: Scan all record occurrences in an area

Problem: List the five credit-hour courses offered by the Mathematics department.

5.3.1.1. Mainframe solution



s	=	ratio of DBC's structure memory access and processing time to mainframe's block access time
	=	0.2
M	=	# of STUDENT records in the data base = 15000
		avg. # of sections per course = 2
		avg. # of courses taken by each student = 4
		avg. # of courses taught by an instructor = 3
N	=	# of COURSE records in the data base = 1000
I	=	# of INSTRUCTOR records in the data base = 700
		length of a STUDENT record = 250 bytes
		length of a CURR-CLASS record = 15 bytes
		length of a SECT-REC record = 30 bytes
		length of a COURSE record = 60 bytes
		length of an INSTRUCTOR record = 250 bytes

avg. # of students per section = $\frac{15000 \times 4}{1000 \times 2} = 30$

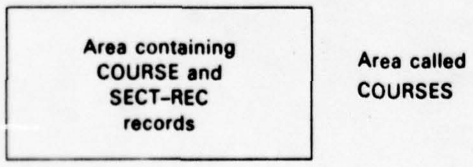
5.3. Record Retrieval

5.3.1. Record Occurrence Scanning

<u>Category</u>: Scan all record occurrences in an area

<u>Problem</u>: List the five credit-hour courses offered by the Mathematics department.

5.3.1.1. Mainframe solution



The entire area containing the COURSE and SECT-REC records is to be scanned sequentially. This is necessary since the COURSE records cannot be directly accessed unless their CALC keys are known. The number of pages to be accessed is the number of occupied by the area. If there are N COURSE records and 2N SECT-REC records, then the execution time in terms of the number of page access is given as

$$\text{Time} = \left\lceil \frac{30 \times 2N + 60N}{p} \right\rceil \simeq 0.5 + \frac{120N}{p}$$

5.3.1.2. DBC with data manipulation language (DML)

All the COURSE records are brought into the host buffer, a cylinder at a time, and checked by the host.

(i) First, a request is sent to the DBC to find the cylinder numbers for the query

(Record-type = COURSE)

This requires structure memory access only.

(ii) For each cylinder number got in (i) a request is sent to the DBC to find records in that cylinder that satisfy the query

(Record-type = COURSE)

This requires no structure memory processing time since cylinder number is already available.

(iii) The predicate (C-CREDITS = 5) is checked by the host itself

$$\begin{aligned}\text{Time} &= S + \left\lceil \frac{60N}{c} \right\rceil m \\ &\simeq 0.2 + \left(\frac{60N}{c} + 0.5 \right) 2.5 \\ &= 1.45 + \frac{150N}{c}\end{aligned}$$

Let N = 1000, p = 448, c = 181440

$$\therefore \text{Performance Ratio} = \frac{0.5 + \frac{120N}{p}}{1.45 + \frac{150N}{c}} = \frac{268.35}{2.28} = 117.7$$

5.3.1.3. DBC with high-level query language

All the COURSE records are directly checked by the DBC with the query

$$(\text{Record-type} = \text{COURSE}) \wedge (\text{C-CREDITS} = 5)$$

$$\begin{aligned} \text{Time} &= S + \left\lceil \frac{60N}{c} \right\rceil m \\ &= 1.65 + \frac{150N}{c} \end{aligned}$$

Notice that the excess time with respect to the DBC with DML is due to the extra structure memory processing. This saves some processing time in the host, but does not show up in the result, since host's processing time is ignored.

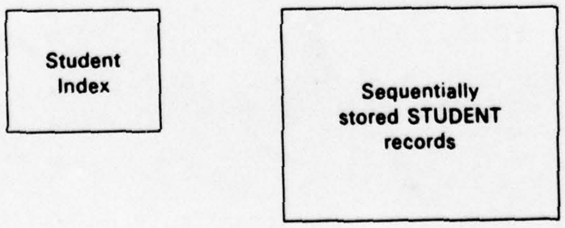
$$\text{Performance Ratio} = \frac{0.5 + \dfrac{120N}{p}}{1.65 + \dfrac{150N}{c}} = \frac{268.35}{2.48} = 108.2$$

5.3.2. Record Type Scanning

Category: • Scan all occurrences of a record type.

Problem: List the students majoring in Computer Science (CS) who have accumulated at least 100 credit-hours and have a grade average of 3.80 or above.

5.3.2.1. Mainframe Solution



Every student is examined via the index. Extra accesses due to overflow are ignored. The number of data pages accessed is:

$$\left\lceil \frac{250M}{p} \right\rceil$$

There is at least one index entry for each data page. Thus, the number of index pages accessed is:

$$\left\lceil \frac{22 \times \left\lceil \frac{250M}{p} \right\rceil}{p} \right\rceil$$

where an index entry is 22 bytes long, 18 for student name and 4 for a pointer.

$$\therefore \text{Time} = \left\{ \frac{1}{p} \left(22\left(\frac{250M}{p} + 0.5\right)\right) + 0.5 \right\} + \left(\frac{250M}{p} + 0.5 \right)$$

$$= \frac{5500M}{p^2} + \frac{250M + 11}{p} + 1$$

5.3.2.2. DBC with DML

All the STUDENT records are brought into the host buffer, a cylinder at a time, and checked by the host. The query used by DBC is

(Record-type = STUDENT)

The predicates are checked by the host, the time for which is not considered here.

$$\text{Time} = S + \left\lceil \frac{250M}{c} \right\rceil m$$

$$\doteq 0.2 + \left(\frac{250M}{c} + 0.5 \right) 2.5$$

$$= 1.45 + \frac{625M}{c}$$

$\therefore$ Performance Ratio = 165.3

5.3.2.3. DBC with query language

All the STUDENT records are completely checked by the DBC. Only the final refined group of records is returned. The query used is

$$\text{(Record-type = STUDENT)} \wedge \text{(ST-MAJOR = CS)} \wedge$$
$$\text{(ST-TOT-CREDITS} \geqslant 100) \wedge \text{(ST-GRADE-AVE} \geqslant 3.80)$$

$$\text{Time} = 4\,S + \left\lceil \frac{250M}{c} \right\rceil m$$

$$= 0.8 + \left(\frac{250M}{c} + 0.5 \right) 2.5$$

$$= 2.05 + \frac{625M}{c}$$

$\therefore$ Performance Ratio = 163.5

5.3.3. Single Record Retrieval

Category: Single record retrieval based on record location mode of direct, calc, or index sequential. (The data base key, calc key, or a value of the indexed attribute is specified.)

Problem: What are the prerequisites for the course CS 311?

5.3.3.1. Mainframe Solution

Since the COURSE record type has a location mode of calc, with course id being the calc key, the address of the necessary record is easily determined. Thus only <u>one</u> page access is required (ignoring overflows). In case the location mode was index sequential, an extra page access would be required for the index.

5.3.3.2. DBC with DML or query language

Query: (Record–type = COURSE) ∧ (C–ID = CS 311)

Since the records in DBC are primarily clustered by record type and secondarily according to location mode, only one cylinder access is required.

Time = 2 s + m = 2.9

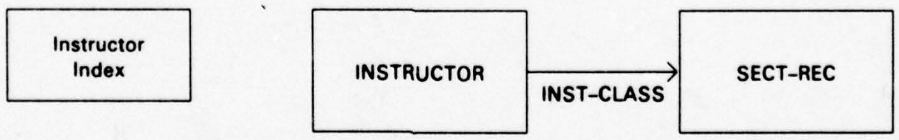
$$\therefore \text{ Performance Ratio } = \begin{cases} \dfrac{1}{2.9} = 0.34, & \text{if location mode is direct or calc} \\ \dfrac{2}{2.9} = 0.69, & \text{if location mode is index seq.} \end{cases}$$

Average = 0.50

5.3.4. Traversing Unclustered Sets–I

<u>Category</u>: Traversing unclustered sets; each set occurrence is small; owner record identified by location mode of direct, calc, or index sequential.

<u>Problem</u>: List the section numbers of the sections taught by White



5.3.4.1. Mainframe solution

One access to the Instructor index. One access to White's Instructor record. If there are n sections taught by White, then three more accesses are required.

Time = 1 + 1 + n

If n=3, then Time=5

5.3.4.2. DBC with DML or query language

(i) Query: (Record-type = INSTRUCTOR) ∧ (I-LAST-NAME = WHITE)

This requires one cylinder access to satisfy, since instructor records are secondarily clustered by last name.

(ii) If the data base key of the above record is d, then another retrieval query is sent.

(Record-type = SECT-REC) ∧ (owner in INST-CLASS = d)

This requires $\left\lceil \frac{2N \times 30}{c} \right\rceil$ accesses since there are N courses, 2 sections per course and 30 bytes per section record.

$$\text{Time} = 4s + \left(1 + \left\lceil \frac{60N}{c} \right\rceil\right) m$$

$$= 0.8 + 2.5 + \left(\frac{60N}{c} + 0.5\right) 2.5$$

$$= 4.55 + \frac{150N}{c}$$

For N = 1000 course,

$$\text{Time} = 4.55 + \frac{150000}{181440}$$

$$= 5.38$$

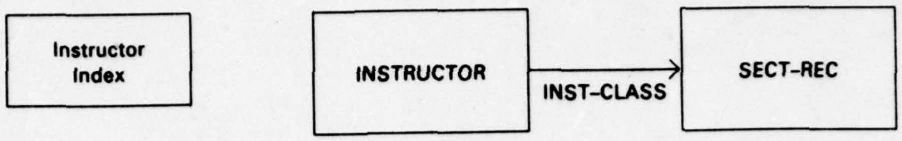
$$\therefore \text{Ratio} = \frac{5}{5.38} = 0.93$$

5.3.5. Traversing Unclustered Sets-II

Category: Traversing unclustered sets; each set occurrence is small; owner records are not identified by location mode.

Problem: What is the average number of sections taught by full professors? (This requires access to all instructor records and then a number of set occurrences of INST-CLASS.)

5.3.5.1. Mainframe solution



Every instructor is examined via the index. The number of data pages accessed for instructors is:

$$\left\lceil \frac{250\,I}{p} \right\rceil$$

There is at least one index entry for each of the above pages. Thus, the number of index pages accessed is:

$$\left\lceil \left(15\mathrm{x} \left\lceil 250I/p \right\rceil \right) /p \right\rceil$$

where an index entry is 15 bytes long, 11 for instructor's last name and 4 for a pointer.

Let there be P professors among all the instructors. If they teach k sections on the average, then a further Pk accesses are required for the section records.

$$\text{Time} = \left\{ \frac{1}{p} \left(15 \left(\frac{250I}{p} + 0.5 \right) \right) + 0.5 \right\} + \left(\frac{250I}{p} + 0.5 \right) + Pk$$

If P=50, k=2, I=700, p=448, then Time=504.2

5.3.5.2. DBC with DML

(i) All the INSTRUCTOR records are brought into the host buffer, a cylinder at a time, and checked by the host. The query used by the DBC is

(Record-type = INSTRUCTOR)

(ii) For each instructor record that qualifies as professor, if its data base key is d, then another query is sent

(Record-type = SECT-REC) ∧ (Owner-in-INST-CLASS = d)

$$\text{Time} \quad = \quad (1 + 2P)\, s + \left(\left\lceil \frac{250\ l}{c} \right\rceil + P \left\lceil \frac{2N \times 30}{c} \right\rceil \right) m$$

where 2N = # of SECT-REC records = 2000
if P=50, l=700, c=181440, s=0.2, m=2.5, then

$$\text{Time} \quad = \quad (1 + 2P)\, 0.2 + \left(\frac{250\ l}{c} + 0.5 + 0.5P + P \frac{60N}{c} \right) 2.5 = 127.7$$

∴ Performance Ratio = 3.9

5.3.5.3. DBC with query language

(i) All the INSTRUCTOR records are checked directly by the DBC using the query

(Record-type = INSTRUCTOR) ∧ (I-POSITION = PROFESSOR)

(ii) For each instructor found in (i), its data base key is loaded in KP. A DBC retrieval query is now used

(Record-type = SECT-REC) ∧ (Owner-in-INST-CLASS ⊂ KP)

$$\text{Time} \quad = \quad 3\, s + \left(\left\lceil \frac{250\ l}{c} \right\rceil + \left\lceil \frac{2N \times 30}{c} \right\rceil \right) m$$

$$= \quad 0.6 + \left(0.5 + \frac{250\ l}{c} + 0.5 + \frac{60N}{c} \right) 2.5$$

$$= \quad 6.33$$

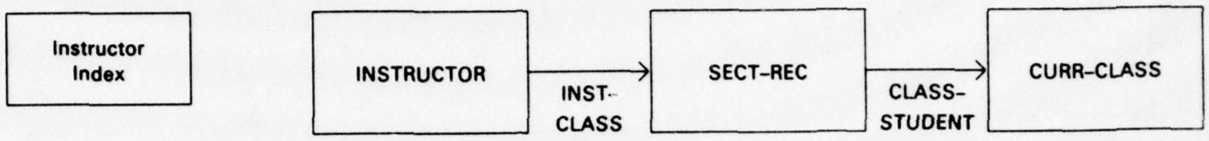
∴ Performance Ratio = 79.7

5.3.6. Traversing Unclustered Sets–III

Category: Traversing unclustered sets; each set occurrence is medium sized; owner record can be identified by location mode of direct, calc or index sequential.

Problem: List the grades obtained by all students in White's classes.

5.3.6.1. Mainframe solution



One index page access is followed by one access to the INSTRUCTOR record of White. If there are n courses taught by White, then n accesses are required to fetch this set of SECT-REC records. If there are x students in each section, then nx accesses are required for the nx CURR-CLASS records.

$$\text{Time} = 1 + 1 + n + nx$$

For $n = 3$ and $m = 30$,

$$\text{Time} = 95$$

5.3.6.2. DBC with DML

(i) Query: (Record–type = INSTRUCTOR) $\wedge$ (I–LAST–NAME = WHITE)

This requires one cylinder access.

(ii) If the data base key of the above record is d, then another retrieval query is sent

(Record–type = SECT–REC) $\wedge$ (owner in INST–CLASS = d)

This requires $\left\lceil \frac{2N \times 30}{c} \right\rceil$ accesses since there are N courses, 2 sections per course and 30 bytes per section record.

(iii) For each of the n records retrieved in step (ii), if its data base key is d, then another retrieval query is sent

(Record-type = SECT-REC) $\wedge$ (owner in CLASS-STUDENT = d)

This requires $n \left\lceil \frac{4M \times 15}{c} \right\rceil$ accesses, since there are M students in the database, 4 courses per student and 15 bytes per CURR-CLASS record

$$\text{Time} = (4 + 2n)\, s + \left(1 + \left\lceil \frac{2N \times 30}{c} \right\rceil + n \left\lceil \frac{4m \times 15}{c} \right\rceil\right) m$$

$$= (4 + 2n)\, 0.2 + \left(1 + 0.5 + \frac{60N}{c} + 0.5n + \frac{60m}{c}\right) 2.5$$

For n = 3 N=1000, m=15000, c=181440, we have

$$\text{Time} = 9.5 + \frac{150\,(N + M)}{c} = 22.7$$

$\therefore$ Performance Ratio = 4.2

5.3.6.3. DBC with query language

(i) Query: (Record-type = INSTRUCTOR) $\wedge$ (I-LAST-NAME = WHITE)

This requires one cylinder access.

(ii) If the data base key of the above record is d, then another retrieval query is sent

(Record-type = SECT=REC) $\wedge$ (owner in INST-CLASS = d)

This requires $\left\lceil \frac{2N \times 30}{c} \right\rceil$ accesses.

(iii) The data base keys of the records found in step (ii) are loaded in KP. Then another query is sent

(Record-type = CURR-CLASS) $\wedge$ (owner in INST-CLASS $\subset$ KP)

This requires $\left\lceil \frac{4M \times 15}{c} \right\rceil$ accesses.

$$\text{Time} = 5s + \left(1 + \left\lceil \frac{2N \times 30}{c} \right\rceil + \left\lceil \frac{4m \times 15}{c} \right\rceil\right) m$$

$$= 1 + \left(1 + 0.5 + 0.5 + \frac{60N}{c} + \frac{60M}{c}\right) 2.5$$

$$= 6 + \frac{150(N + M)}{c}$$

For N = 1000, M = 15000, c = 181440, we have

Time = 19.2

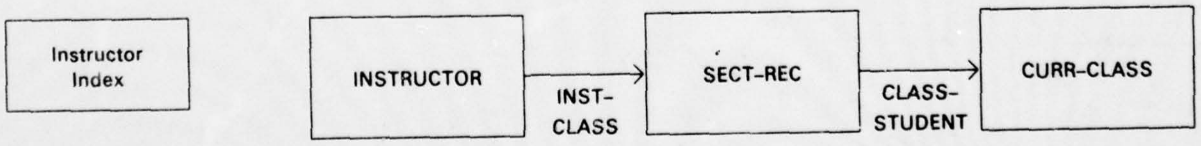
$\therefore$ Performance Ratio = 4.9

5.3.7. Traversing Unclustered Sets–IV

Category: Traversing unclustered sets; each set occurrence is medium sized; owner records are not identified by location mode.

Problem: Are the professors older than 50 really stingy about grades, or are they quite liberal. (We need to find the average grade over all students taught by professors born before 1928.)

5.3.7.1. Mainframe solution



Each instructor is examined via the index. The number of data pages

Each instructor is examined via the index. The number of data pages accessed for instructors is:

$$\left\lceil \frac{250\,I}{p} \right\rceil$$

There is at least one index entry for each of the above pages. Thus, the number of index pages accessed is:

$$\left\lceil \left(15\times \left\lceil 250l/p \right\rceil \right)/p \right\rceil$$

where an index entry is 15 bytes long, 11 for instructor's last name and 4 for a pointer.

Let there be <u>P professors</u> older than 50. If they teach <u>k sections</u> on the average, then a further Pk accesses are required for the SECT-REC records.

Let there be <u>y students</u> per section. Then a further Pky accesses are required for the CURR-CLASS records.

$$\text{Time} = \left\{ \frac{1}{p} \left(15 \left(\frac{250l}{p} + 0.5 \right) \right) + 0.5 \right\} + \left(\frac{250l}{p} + 0.5 \right) + Pk + Pky$$

If P = 30, k = 2, l = 700, p = 448 and y = 30, then Time = 2264.7.

5.3.7.2. DBC with DML

(i) All the INSTRUCTOR records are brought into the host buffer, a cylinder at a time, and checked by the host. The query used by the DBC is

(Record-type = INSTRUCTOR)

This requires $\left\lceil \dfrac{250\,l}{c} \right\rceil$ cylinder accesses.

(ii) For each instructor that qualifies as professor over 50 years old, if his data base key is d, then another query is sent

(Record-type = SECT=REC) $\wedge$ (owner in INST-CLASS = d)

This requires $p \left\lceil \dfrac{2N \times 30}{c} \right\rceil$ cylinder accesses, since p professors qualify, N courses in the database, 2 sections per course and 30 bytes per SECT-REC record

(iii) For every record got in step (ii), if its data base key is d, then another query is sent

(Record-type = CURR-CLASS) $\wedge$ (owner in CLASS-STUDENT = d)

This requires $Pk \left\lceil \frac{4M \times 15}{c} \right\rceil$ accesses, if there are k sections per professor, M students in the database, 4 courses per student and 15 bytes per CURR-CLASS record. (Note that steps (i), (ii), and (iii) are actually interleaved, but for timing considerations they can be considered separate).

$$\text{Time} = (1 + 2p + 2pk)\, s + \left(\left\lceil \frac{250\, l}{c} \right\rceil + P \left\lceil \frac{60N}{c} \right\rceil + Pk \left\lceil \frac{60M}{c} \right\rceil \right) m$$

$$= (1 + 2p + 2pk)\, 0.2 + \left(0.5 + 0.5p + 0.5pk + \frac{250\, l}{c} + \frac{60NP}{c} + \frac{60MPk}{c} \right) 2.5$$

For P=30, k=2, l=700, c=181440, N=1000, m=15000, we have

Time = 921.2

$\therefore$ Performance Ratio = 2.46

5.3.7.3. DBC with query language

(i) All the INSTRUCTOR records are checked directly by the DBC using the query

(Record-type = INSTRUCTOR) $\wedge$ (I-POSITION=PROFESSOR) $\wedge$ (I-BYEAR < 1928)

This requires $\left\lceil \frac{250\, l}{c} \right\rceil$ cylinder accesses.

(ii) For each instructor record found in (i), its data base key is loaded in KP. A retrieval query is now used

(Record–type = SECT=REC) ∧ (owner in INST-CLASS ⊂ KP)

This requires $\left\lceil \frac{2N \times 30}{c} \right\rceil$ cylinder accesses, where there are N courses, 2 sections per course and 30 bytes per SECT-REC record

(iii) For each record found in (ii), its data base key is loaded in KP. A retrieval query is now used

(Record–type = CURR-CLASS) ∧ (owner in CLASS-STUDENT ⊂ KP)

This requires $\left\lceil \frac{4M \times 15}{c} \right\rceil$ cylinder accesses, where there are M students in the database, 4 courses per student and 15 bytes per CURR-CLASS record

$$\text{Time} = 5\,s + \left(\left\lceil \frac{250\,I}{c} \right\rceil + \left\lceil \frac{60N}{c} \right\rceil + \left\lceil \frac{60M}{c} \right\rceil \right) m$$

$$= 1 + \left(1.5 + \frac{250\,I}{c} + \frac{60N}{c} + \frac{60M}{c} \right) 2.5$$

$$= 4.75 + \frac{625\,I}{c} + \frac{150(N+M)}{c}$$

For I=700, N=1000, M=15000, c=181440, we have

Time = 20.4

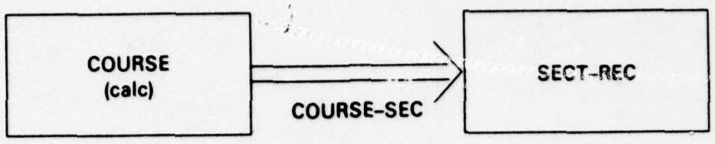
∴ Performance Ratio = 111.0

5.3.8. Traversing Clustered Sets–I

Category: Traversing clustered sets; each set occurrence is small; owner record is identified by location mode of direct, calc or index sequential.

Problem: List the sections of the course CS 611.

5.3.8.1. Mainframe solution



One access for the page containing the required COURSE record (because location mode of COURSE is calc with course number being the calc key). The same page contains the corresponding section records, since the COURSE-SEC set is clustered.

Time = 1 , if location mode is direct or calc

= 1+1, if location mode is index sequential

5.3.8.2. DBC with DML or query language

(i) Query: (Record-type = COURSE) $\wedge$ (C-ID = CS 611)

This requires one cylinder access, because of primary clustering by record type and secondary clustering by course id.

(ii) If the data base key of the above record is d, then another query is sent

(Record-type = SECT-REC) $\wedge$ (Owner-in-COURSE-SEC = d)

This requires one cylinder access, because of secondary clustering of SECT-REC records by the owner identifier in the COURSE-SEC set.

Time = 4s + 2m

= 0.8 + 5

= 5.8

$$\text{Performance Ratio} = \begin{cases} \dfrac{1}{5.8} = 0.17, \text{ if loc. mode is direct or calc} \\ \dfrac{2}{5.8} = 0.34, \text{ if loc. mode is index sequential} \end{cases}$$

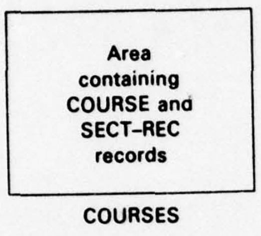
$$\text{Average} = \frac{1.5}{5.8} = 0.26$$

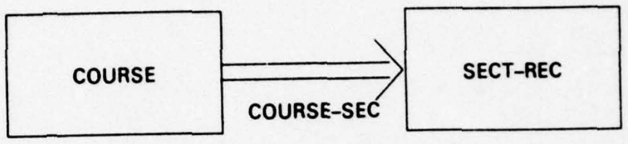
5.3.9. Traversing Clustered Sets–II

Category: Traversing clustered sets; each set occurrence is small; owner records are not identified by location mode.

Problem: Find out if in any Computer Science graduate section, the number of students registered exceeds the predetermined absolute limit. (Note: Course number of a Computer Science graduate section is CS 600 through CS 999.)

5.3.9.1. Mainframe solution





The area called COURSES has to be completely traversed in order to identify the Computer Science graduate courses. If there are N courses and 2N sections, then this requires the accessing of:

$$\lceil (N \times 60 + 2N \times 30)/p \rceil \quad \text{pages.}$$

Having obtained an appropriate COURSE record, no further accesses are required for the corresponding SECT–REC records since the COURSE–SEC set is clustered.

$$\text{Time} = 0.5 + \frac{120N}{p}$$

For N = 1000 and p = 448, we have Time = 268.4.

5.3.9.2. DBC with DML

(i) All the COURSE records are brought into the host buffer, a cylinder at a time, and checked by the host. The query used by the DBC is

(Record-type = COURSE)

This requires $\left\lceil \frac{N \times 60}{c} \right\rceil$ cylinder accesses.

(ii) If there are z courses that qualify as graduate courses in Computer Science, then for every such course, if its data base key is d, another query is sent

(Record-type = SECT-REC) $\wedge$ (Owner-in-CLASS-SEC = d)

This step requires z cylinder accesses.

$$\begin{aligned}
\text{Time} &= (1 + 2z)\,s + \left(\left\lceil \frac{60N}{c} \right\rceil + z\right) m \\
&= (1 + 2z)\,0.2 + \left(0.5 + \frac{60N}{c} + z\right) 2.5 \\
&= 1.45 + 2.9z + \frac{150N}{c}
\end{aligned}$$

For N=1000, z=30 and c=181440, we have

Time = 89.3

$\therefore$ Performance Ratio = 3.0

5.3.9.3. DBC with query language

(i) Query: (Record-type = COURSE) $\wedge$ (CS 600 $\leqslant$ C-ID $\leqslant$ CS 999)

This requires $\lceil N \times 60/c \rceil$ cylinder accesses, there being N courses in the database.

(ii) Optimization

(a) A request is sent to DBC to determine the number of cylinders to be searched in satisfying the query

(Record–type = SECT–REC)

(b) If the above number is less than the number of records found in step (i), then the data base keys of the records found in step (i) are loaded in KP and a query is sent

(Record–type = SECT–REC) $\wedge$ (Owner–in–COURSE–SEC $\subset$ KP)

(c) Otherwise, for each COURSE record found in step (i), if its data base key is d, then a query is sent

(Record–type = SECT–REC) $\wedge$ (Owner–in–COURSE–SEC = d)

If there are z courses that qualify, and there are 2N sections, then

$$\text{Time} = \begin{cases} 4\,s + \left(\left\lceil \frac{60N}{c} \right\rceil + \left\lceil \frac{2N \times 30}{c} \right\rceil \right) m, & \text{if } z \geq \left\lceil \frac{2N \times 30}{c} \right\rceil \\ (3 + 2z)\,s + \left(\left\lceil \frac{60N}{c} \right\rceil + z \right) m, & \text{otherwise} \end{cases}$$

For z = 30, N=1000, c=181440, we have

$$\text{Time} = 4 \times 0.2 + \left(1 + \frac{60N}{c} + \frac{60N}{c} \right) 2.5$$

$$= 3.3 + \frac{300N}{c}$$

$$= 5.0$$

$\therefore$ Performance Ratio = 53.7

CATEGORIES LISTING

1. Scan all record occurrences in an area.
2. Scan all occurrences of a record type.
3. Retrieve a single record (or duplicate) based on record location mode of direct, calc or index sequential.
4. Traverse unclustered sets; each set occurrence is small; owner record identified by location mode of direct, calc or index sequential.
5. Traverse unclustered sets; each set occurrence is small owner records not identified by location mode.
6. Traverse unclustered sets; each set occurrence is medium-sized; owner record identified by location mode of direct, calc or index sequential
7. Traverse unclustered sets; each set occurrence is medium-sized owner records not identified by location mode.
8. Traverse clustered sets; each set occurrence is small; owner record identified by location mode of direct, calc or index sequential.
9. Traverse clustered sets; each set occurrence is small; owner records not identified by location mode.

Table 5-1. CATEGORIES LISTING

Performance Ratio / Category →		1	2	3	4	5	6	7	8	9
Using DML	Page Size = 112 bytes	470.1	754.1	0.50	0.93	14.7	4.2	3.9	0.26	12.0
	Page Size = 448 bytes	117.7	165.3	0.50	0.93	3.9	4.2	2.5	0.26	3.0
Using Query Language	Page Size = 112	432.2	745.7	0.50	0.93	295.8	4.9	178.1	0.26	214.4
	Page Size = 448	108.2	165.3	0.50	0.93	79.7	4.9	111.0	0.26	53.7

Table 5-2. RECORD RETRIEVAL PERFORMANCE RATIOS

Time	Category →	1	2	3	4	5	6	7	8	9
Page Size = 112 Bytes	Mainframe	1071.9	40060.0	1.5	5.0	1872.3	95.0	3632.8	1.5	1071.9
	DBC (using DML)	2.3	53.1	2.9	5.4	127.7	22.7	921.2	5.8	89.3
	DBC (using query language)	2.5	53.7	2.9	5.4	6.3	19.2	20.4	5.8	5.0
Page Size = 448 Bytes	Mainframe	268.4	8782.6	1.5	5.0	504.2	95.0	2264.7	1.5	268.4
	DBP (using DML)	2.3	53.1	2.9	5.4	127.7	22.7	921.2	5.8	89.3
	DBP (using query language)	2.5	53.7	2.9	5.4	6.3	19.2	20.4	5.8	5.0

Table 5-3. ABSOLUTE TIMES IN RECORD RETRIEVAL

5.4. Offloading of the host

There can be a considerable saving in CPU (host) time when the DBC is used. This is due to the fact that in the absence of the DBC, all predicates are evaluated in the CPU.

5.4.1. Example

For example, consider the following request:

List the students majoring in Computer Science (CS) who have accumulated at least 100 credit–hours and have a grade average of 3.80 or above.

5.4.1.1. Mainframe solution

(i) Every STUDENT record is stored, one at a time, in the host's work area. A STUDENT record consists of 250 bytes. Assuming that 4 bytes can be stored at a time irrespective of field size, then a single record requires storing 63 groups of 4 bytes. Let each store operation require

(a) a load instruction for loading from buffer page to accumulator

(b) a store instruction for storing from accumulator to work area

(c) an instruction to increment work area address in an index register

(d) an instruction to increment page offset in an index register

(e) a compare and branch instruction for loop termination

Then there are at least 5 instructions executed per 4 bytes of a record.

∴ Total # of instructors executed in storing a single record = 5 x 63 = 315

If there are 10000 students in the data base, then the total number of instructions executed in storing all students = 3,150,000.

(ii) Every STUDENT record is now checked against one or more of the predicates p_1, p_2, p_3, where

p_1 is (ST–MAJOR = CS)

p_2 is (ST–TOT–CREDITS $\geqslant$ 100)

p_3 is (ST–GRADE–AVE $\geqslant$ 3.80)

Assume that 2% of all students satisfy p_1. 1% of all students satisfy p_1 and p_2. 0.1% of all students satisfy p_1 and p_2 and p_3. To evaluate a single predicate requires the following three instructions.

(a) a load instruction

(b) a compare instruction to evaluate the predicate

(c) a branch instruction

The total number of predicates to be evaluated for all student records is

$$10000 + 2\% \times 10000 + 1\% \times 10000 = 10300$$

Thus, the number of instructions executed = 30900.

Total for steps (i) and (ii)

= 3,180,900 instructions.

5.4.1.2. DBC solution

If a query language is used, then all predicates are evaluated by the DBC and only the refined list of students is given back to the host.

$\therefore$ Total # of students sent to host

$$= 10000 \times 0.1\% = 10$$

Storing each student record requires 315 instructions. Thus, total number of instructions executed by host

= 3150.

$$\therefore \frac{\text{Time spent by host in storing and checking records without the DBC}}{\text{Time spent by host in storing and checking records with the DBC}}$$

$$= \frac{3180900}{3150} = 1010$$

Notice that most of the time is spent by the CPU (without the DBC) in storing and checking records that are not needed in the final output set. In general, if the DBC can be used to evaluate predicates, then 90% of CPU time can be saved, assuming that 90% of all retrieved data in a conventional system is actually not a part of the final output set.

5.5. Record Update

Assumptions

1. Accesses required for storage management are ignored.

2. Two directory attributes per record; and updates to Structure Memory require no extra time.

3. Owner pointers are there in each member record of a set.

4. u = Ratio of update time of a cylinder to a page access time

= 0.5 latency + 1 transfer (for read) + 1 processing + 0.5 latency + 1 transfer (for write)

= 4

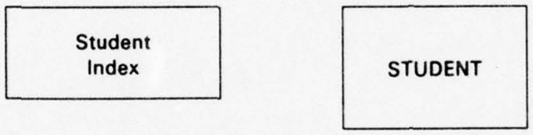
5.5.1. Insertion - I

Category: Load a record into the data base.

Problem: Add another STUDENT record occurrence.

5.5.1.1. Mainframe solution

Mainframe Solution



At least one access, possibly followed by an update, of the student index. The page in which the STUDENT record must be loaded is found, accessed, updated and stored back. During the loading, a data page may already be full, thus necessitating another data page access. But we ignore the chances of overflow.

Time = read an index page + write an index page + read a data page + write a data page

= 4

5.5.1.2. DBC solution

Two structure memory accesses for the two directory keywords.

One cylinder access and update

Time = 2s + 1n

= 0.4 + 4 = 4.4

Performance Ratio = $\frac{4}{4.4}$ = 0.91

5.5.2. Insertion - II

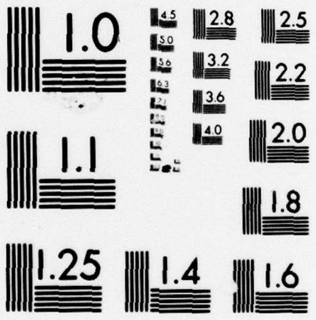
Category: Insert a record into a set.

Problem: Assign section number 5 of the course CS 610 to the instructor White.

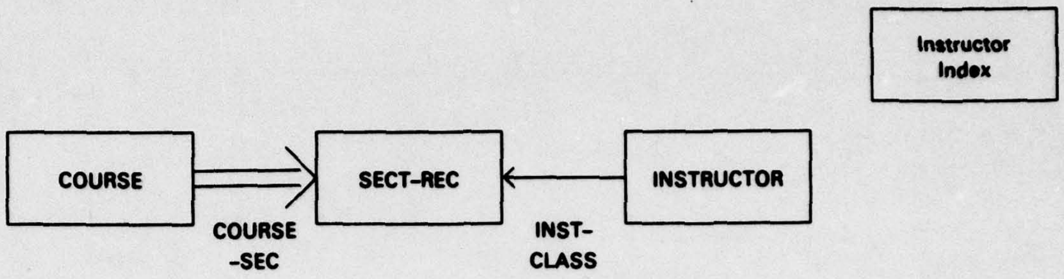
AD-A082 074 SPERRY UNIVAC ROSEVILLE MN F/G 9/2
DATA BASE COMPUTER RESEARCH.(U)
NOV 79 H A FREEMAN, J BANERJEE, R B BATMAN N00014-78-C-0487
UNCLASSIFIED TMA-00789 NL

2 OF 2
AD-
A082074

END
DATE
FILMED
4-80
DTIC


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

5.5.2.1. Mainframe solution

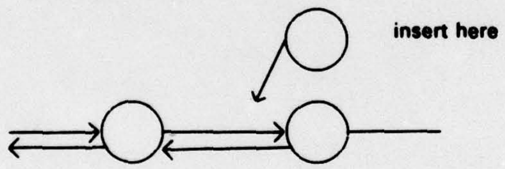


One page access for the course record. In the same page, the required section (SECT-REC) record is also found. This record is held in storage for updating pointers and its address is noted.

One index page access for instructor record's index.

One page access for the INSTRUCTOR record itself.

One-half of the set occurrence of INST-CLASS with White as owner is now traversed (on the average) to find the logical position of the section record to be inserted. If there are n sections per instructor, then n/2 page accesses are required to find position. Three pages require to be updated.



∴ Time = 1 course page + 1 index page + 1 instructor page + n/2 section pages + 2 x 3 section pages

or Time = 9 + n/2

For n = 4,

Time = 11

5.5.2.2. DBC Solution

(i) Query: (Record-type = INSTRUCTOR) ∧ (I-LAST-NAME = White)

This retrieval query requires 1 cylinder access.

(ii) If the data base key of this record is d, then send a record modification query to modify the record satisfying

(Record-type = SECT-REC) $\wedge$ (SEC-NUM = 5)

$\wedge$ (Owner-in-COURSE-SEC = CS 610)

by adding the keyword (owner-in-INST-CLASS, d). This requires 1 cylinder access for modification.

Time = 5s + 1m + 1u

= 5 x 0.2 + 1 x 2.5 + 1 x 4

= 7.5

Performance Ratio = 11/7.5 = 1.47

5.5.3. Deletion - I

Category: Remove a record from a set.

Problem: Relieve White from the responsibility of teaching section number 5 of the course CS 610.

The analysis is the same as in the case of the assignment of the section to White.

5.5.3.1. Mainframe solution

Time = 11

5.5.3.2. DBC solution

Time = 7.5

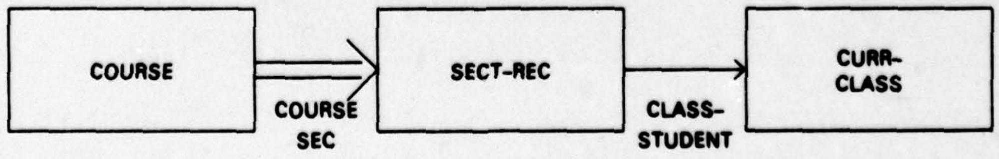
Performance Ratio = 11/7.5 = 1.47

5.5.4. Deletion - II

Category: Delete records from the data base.

Problem: CS 611 was offered in the last quarter. But its section records are not valid any more at the end of the quarter. Furthermore the CLASS-STUDENT set occurrences belonging to these SECT-REC records must be removed.

5.5.4.1. Mainframe solution



One page access for the required course records.

If there are k sections per course, then all k SECT-REC records are in the same page. But this page needs to be updated. Thus, one more access for writing back the page.

If there are y students per section, then ky page accesses are required for reading and ky page accesses are required for writing, in the processing of removing CURR-CLASS records from the CLASS-STUDENT set.

$$\text{Time} = 2 + 2ky$$

For $k = 2$ and $y = 30$, we have

$$\text{Time} = 122$$

5.5.4.2. DBC Solution

(i) One cylinder requires update; the one which contains records satisfying the query

(Record-type = SECT-REC) ∧ (Owner-in-COURSE-SEC = CS 611)

(ii) The data base keys of the above n records are loaded in KP and cylinders are $\left\lceil \frac{4M \times 15}{c} \right\rceil$ updated — those containing records satisfying the query.

(Record-type = CURR-CLASS) ∧ (Owner-in-CLASS-STUDENT ⊂ KP)

Note that M is the number of students in the data base, each student takes four courses and each CURR-CLASS record is 15 bytes long.

$$\begin{aligned}\text{Time} &= 3s + \left(1 + \left\lceil \frac{4M \times 15}{c} \right\rceil\right) n \\ &= 0.6 + \left(1 + 0.5 + \frac{60M}{c}\right) 4 \\ &= 6.6 + \frac{240M}{c}\end{aligned}$$

For $M = 15000$ and $c = 181440$, we have

$$\text{Time} = 19.8$$

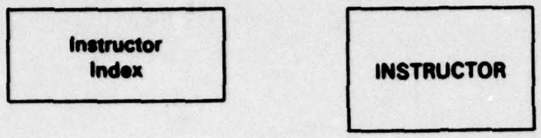
$\therefore$ Performance Ratio = 6.2

5.5.5. Modification

<u>Category</u>: Modify records.

<u>Problem</u>: Allocate a 5% increment in salary to all instructors in Computer Science.

5.5.5.1. Mainframe solution



Since the instructor records are not clustered by department, we may assume that all pages contain an instructor record of concern (i.e., of Computer Science).

Thus, $\left\lceil \frac{250\ I}{p} \right\rceil$ data pages for instructors are read and written back.
If there is at least one index entry for each data page, then a total of:

$$\left\lceil (15 \times \left\lceil 250\ I/p \right\rceil)/p \right\rceil$$ index pages are to be read.

$$\text{Time} = \left\{ \frac{1}{p} \left(15 \left(\frac{250\ I}{p} + 0.5 \right) \right) + 0.5 \right\} + 2 \left(\frac{250\ I}{p} + 0.5 \right)$$

If I = # of instructors = 700 and p = page size = 448 bytes then

Time = 795.8

If p = 112 bytes, then

Time = 3335.8

5.5.5.2. DBC solution

All cylinders containing INSTRUCTOR records are to be updated. The query to be used to identify records is

$$(\text{Record-type} = \text{INSTRUCTOR}) \wedge (\text{I-DEPT} = \text{CS})$$

$$\text{Time} = 2s + \left(\left\lceil \frac{250\,l}{c} \right\rceil\right) u$$

$$= 0.4 + 4\left(0.5 + \frac{250\,l}{c}\right)$$

For l = 700 and c=181440, we have

$$\text{Time} = 2.4 + \frac{1000\,l}{c}$$

$$= 6.3$$

$$\therefore \text{Performance Ratio} = \begin{cases} 126.3 \text{ for } p = 448 \text{ Bytes} \\ 529.5 \text{ for } p = 112 \text{ Bytes} \end{cases}$$

6. DATA BASE COMPUTER ACCESS CONTROL

6.1. Introduction

This section discusses how access control mechanisms could be implemented in an environment using a Data Base Computer. The discussion is limited to the implementation of access control constraints defined in Logical Data Models supported by the Data Base Computer – that is to say that there is no discussion of possible access control constraints which could be developed in DBC software and then utilized to provide enhanced access control mechanism in supported Logical Data Models.

The parameters which influence permission or denial of access are first described. Then, examples of constraints involving those parameters are given. Types of constraints are then classified as more or less suitable for implementation in DBC software as opposed to "host" or Logical Data Manager Software. Techniques for implementation in DBC software are discussed along with their advantages and disadvantages.

6.2. Access Control Parameters

In determining whether or not to grant permission for a particular data request a data base management system (DBMS) evaluates an access control constraint with regard to particular parameters defining the request. The parameters can be classified into the following three major types – the user, the operation, the object.

6.2.1. User

The user is the originator of the request. It is assumed that the operating system handles the identification and if necessary the authentication of the user and possibly assigns the user to a user group which associates the user with certain access rights. These parameters can be considered as intrinsic or direct attributes of the user.

Circumstantial or "indirect attributes"[13], of the user may be such parameters as password (s) furnished by the user or his program, his terminal–id , the time of day.

Effectively the user parameters are the credentials provided by the user to support his claim to data base access.

6.2.2. Operation

The operation is the type of manipulation which the user is attempting to perform in the request. It will orginate in the form of a DML request appropriate to the particular Logical Data Model but will be related to the basic operations of read/insert/delete/ update. However, a higher level of operation can be relevant. This would normally be a privileged program which may be allowed to perform basic operations but only for a specific purpose. Only data transfer operations are discussed here but similar considerations hold for operations such as opening files.

6.2.3. Object

The object describes the data values or relationships involved in the request. The data objects involved are defined in terms of the Logical Data Model (e.g. records, sets, relations, segments, attributes) by type and the particular data occurrence (s) are defined by selection criteria (e.g. record selection expression, predicate).

6.3. Access Control Constraints

Various types of access control constraint can be specified in terms of the parameters defined in section 2. Examples of these are given by date[14] from which some of the following are derived:–

1. a user may read, for any employee, the employee number, name, address, and department number, but may update only the name and address values ;
2. a user may read, for any employee, the employee number and salary, and may update the salary value but only between 09.00 and 17.00 and only from a terminal located in the payroll office ;
3. a user may read, for an employee, the employee number and assessment, but only if the user is the manager of the department identified by the department number value for that employee ;
4. a user may alter the salary value for any employee but only if the current and new salary values are less than 5000 ;
5. a user may generate certain reports summarizing salary values, but may not access individual salary values ;
6. a user may update the address values for employees but only if the appropriate password has been provided by the user ;
7. the values of employee number, name, and address for same employees can be accessed by only one particular user.

These types of constraints can be usefully categorized as either uniform within a data item type or varying for different occurrences within a data item type. These two categories can be called data value independent and data value dependent. Thus the above examples are categorized as follows:

1. independent
2. independent
3. dependent
4. dependent
5. independent
6. independent
7. dependent

6.4. Access Control Mechanisms

Access control Mechanisms are the implementation of the constraints discussed previously. The section examines which constraints could most advantageously be implemented in DBC software rather than Logical Data Manager (LDM) software.

The principal justification of the DBC is its superior capability to accelerate access to selected data item occurrences. Therefore the DBC would seem to be a strong candidate for implementation of data value dependent constraints – for instance examples 3, 4 and 7. In examples 3 and 4 the access constraint can be implemented as an additional selection criterion on the data values/relationships. Example 7 could be implemented in a similar way if the permitted user identifications were held as data in the data base related to the object data. Note that both 3 and 7 would require the DBC to perform comparisons between the requesting user identification and permitted user identifications – which is an operation suited to the DBC.

The henefit of implementing data value dependent constraints in the DBC arises from the fact that both constraint evaluation and data access have to take place. In data value independent constraints this is not the case since permission/denial can be evaluated without data access. It is therefore possible to avoid unnecessary traffic between LDM and DBC when permission is denied. Conversely if the DBC were to implement such constraints then the LDM would need to supply the DBC with the necessary access control parameters (e.g. terminal–id, time–of–day, in example 2.

To confirm the suitability of the DBC for implementation of data value dependent constraints it will be necessary to investigate the following areas:

1. what is involved for different LDM's to transform a value dependent constraint, expressed in terms of the appropriate Logical Data Models, into a form compatible with the Data Storage Language (DSL) interface? An objective should be to encode the access control parameters into an access control "addendum" to the original DSL request, and then to satisfy this "addendum" by means of values pre–existing in the data base or values introduced into the data base specifically to represent an access control constraint. Thus the DBC selection mechanism should be common to both the normal and the access control components of the DSL request.

2. how will it be possible for the LDM to distinguish access control exception conditions from other (e.g. no qualifying data) exceptions? This may be necessary for purposes such as "threat monitoring" or to enable evasive action (e.g. disconnect and dial back) to be initiated. A solution may be for the DBC to perform access control DSL selection in a seperate pass on the abstract set derived from the first DSL pass.

3. how should data value dependent constraints involving CODASYL type data base procedures be implemented? This type of situation is currently not very common as most CODASYL implementations do not permit data base procedures to perform data base access or indeed to access the object record. Thus data base procedure constraints are usually data value independent (e.g. example 2). However in value dependent cases the solutions are either to pass back occurrences to the LDM for data base procedure invocation: or to implement the data base procedure in the DBC. Neither of which are entirely desirable. Legislation to eliminate the need for data base procedures by allowing more powerful declarative specifications may be the saviour.

4. it will be necessary for the DBC to be aware of user identities, or, more likely, user group identities. These identities will appear in the DSL access control addendum to be validated against values stored in the data base identifying which user groups are permitted/denied access to certain objects, for certain operations (examples 3 and 2 in section 3). Co–ordination and maintenance of user–ids with one or more separate hosts with potentially different LDM's may require sophisticated data base administrator utilities.

6.5. Access Control Conclusions

Definite advantages seem to accrue from implementing data value dependent access control constraints in the Data Base Computer as opposed to the Logical Data Manager. These arise from the search, extract, compare nature of this type of constraint.

No particular advantage appears to lie in implementing data value independent constraints in this way. If these are implemented in the Logical Data Manager then transfer of DSL requests to the Data Base Computer can be avoided, in those cases where permission is denied.

The Logical Data Managers will require the capability to translate value dependent constraints into DSL request format. Also some value dependent constraints will require supporting data structures to be established in the data base to enable the constraint to be applied at the occurrence level.

Particular attention needs to be paid to the areas of access control exception signaling and of CODASYL style data base procedures.

7. SECURITY CONSIDERATIONS

7.1. Definitions and Scope (Security Concepts)

Security encompasses the ability of a system to properly preserve the confidentiality, integrity, and availability of information in the face of any and all credible threats according to an established security policy. For the most part, preservation of integrity and availability are already general goals and will not be covered here, except that the manner in which a security policy is expressed by users and administrators includes both who (or what) has the right to read specific pieces of information and who has the right to write or alter specific pieces of information. Since a DBC system itself cannot be directly programmed, the major threats it must defend against are explicit requests for unauthorized information or actions, any errors it can be induced to make via the interface to a host computer, human errors, and subversion or error in the design and implementation of the system itself. A Stand–aline DBC system has a slightly broader set of threats to defend against since it is more directly accessible.

The most general form of security policy to be supported by a DBC system has both a mandatory and a discretionary aspect. The mandatory aspect is expressed in terms of security labels applied to people (or parts of the system acting on their behalf) and to data. The labels are partly expressed by individual users, partly implied by the history of activity, but mostly expressed by security administrators.

In general, an individual has no control over who can access data under a mandatory policy and the system must enforce that. In particular, it must not be possible – without special permission – for a person who has access to information under the mandatory policy to allow someone else who is not so authorized, to access it. The discretionary aspect provides an additional refinement where individual users (or administrators) can selectively permit or deny access to other people, once the mandatory policy requirements permitting the access have been fulfilled.

The labeling provided under the mandatory policy allows information to be given a level of sensitivity and to be segregated into one or more compartments or categories of information. A given piece of information is of course of only a single level of sensitivity, but it may belong to many compartments or categories. People, in turn, are authorized to access information up to a given level of sensitivity, but no higher, but they may be authorized to access information in many categories or compartments. For an access to be permitted, the accessor must have permission at least as high as the level of sensitivity of the information and he must be permitted to access information in all of the categories it resides in (is labelled with). (He of course may have much broader permission than that.)

Generally speaking, the accessors of information are called subjects and the pieces of information itself are called objects. In a computer system the true accessors (subjects) are of course not people, but parts of the system and it must be generally assured that these subject have no higher effective permission than the people they are surrogates for. When a given host cannot, for instance, be assured to properly identify its subjects, the host as a whole must be assumed to have a "lowest common denominator" permission that will be enforced by the DBC regardless of what the host says to it.

7.2. General Goals

A DBC system eventually should support a mandatory security policy containing both security levels and security categories ; at most 8 levels and 256 categories need to be supported on a given system. Any given piece of data is on the average in no more than a half-dozen categories, and any person is on the average authorized for no more than a dozen categories, but these typical valves should not limit system maxima.

It should also support a discretionary policy based on lists of subject i.d.'s, and on named groups of subjects. The policy should be enforceable at all levels of data structure in the structured data memory, i.e., from areas (files), to records, to individual fields (items) in records. Security attributes may also be specified for particular physical media – e.g., disks or tracks. The security attributes of data should either apply to all instances of a particular data structure (e.g., all instances of a particular record type or item) or they can vary from instance to instance. The system should eventually be capable of being certified to meet the most stringent requirements of the U.S. Federal Government, in particular those of the Department of Defense. The system should also support CODASYL privacy locks and keys down to the item level ; both explicit keys (passwords) and arbitrary decision procedures should be supported when the attached host computer supports them. (Some forms of the decision procedures should be capable of being executed or interpreted by the DBC system itself.) The security provided by subschemas should also be supported. Requirements for controls over statistical data bases are to be determined.

7.3. System Environment

A DBC system should support the above security goals to the best extent possible even when it is connected to hosts that are less secure than it is. In particular, multiple hosts, each internally insecure but dedicated to processing information at or below a given security level or category set, should be able to be connected to a single DBC system, with the DBC system properly enforcing a mandatory security policy between the hosts. In such a case, one of the hosts (which could be a dedicated mini–computer) will act as the security administrator's host. When a DBC system is directly connected to a distributed processing or communications network, appropriate security should continue to be enforced.

7.4. Security Administration

Suitable interfaces for security administration should be provided. In a DBC system this includes mostly the ability to generate suitable audit trails and to generate, inspect, and alter security profile information. (Probably only via explicit requests from a host.) In a Stand–alone DBC system, specific well–human–engineered interfaces for security administrators should be provided. Security administration should be capable of being delegated and distributed, even across a distributed processing communications network. Security administration should be capable of being separated from data base administration.

7.5. System Tailoring

The most general form of DBC security need not be present in all systems. In particular, any given site may elect not to use any or all of the forms of policy, <u>in toto</u> or for specific collections of information. Storage and performance overhead should be suitably diminished or removed when any given feature is not used. In particular, there should be very little overhead due to security when no access controls are enforced below the file (or area) level or when the security profiles are invariant across all instances of a given set, record, or item type. Accurate system models should be developed to allow forecasting of the performance and storage costs associated with electing to use any particular form of policy or granularity of access control.

7.6. Host System Compatibility

When used to replace or augment existing mass storage, either as part of a general-purpose file system or as part of a current DBC system, it should be capable of enforcing the kind and degree of security enforced by host system in a user and system administrator transparent way, even though that may be a variance with the general goals for a DBC.

7.7. Implementation Considerations

It is desirable that all security information (such as classifications, clearances and discretionary access control lists) be able to be stored as part of the DBC, in a form it can access directly tc make security decisions, but this may require either duplicate storage of the information (since the host already maintains some of it in a most likely different form) or significant changes to host software.

Internally, the DBC system should provide integrity protection of its meta data--bases, programs, and working storage. It would be desirable that the internal processing elements have rudimentary forms of addressing protection and privilege state control to limit the amount of damage done by hardware or software errors. The manner of representing security information, especially for item dependent profiles, needs further thought to reduce storage costs and overhead. It is desirable that security profile information for an object be kept physically separate from the object but logically coupled to it. A general unanswered question is whether the true security decisions are made by the host computer on information it requests from the DBC, or whether all decisions are made by the DBC itself, based on little or no information from the host. (The latter is necessary if the host cannot be trusted sufficiently, as when multiple insecure hosts are connected to a single DBC system. An interesting special case of this is when multiple virtual machines in a given host communicate with the same DBC system.)

8. REFERENCES

1. Computer Weekly, No. 573, Oct. 27, 1977, p. 1.

2. Datamation, Vol. 24, No. 5, May 1978, p. 288.

3. Copeland, G.P. Jr., et al, "The Architecture of CASSM: A Cellular System for Non-numeric Processing," Proceedings of the First Workshop on Computer Architecture, 1973, pp. 121-128.

4. Ozkarahan, E.A., et al, "RAP - An associative computer for data base management," Proceedings National Computer Conference, 1975, pp. 379-387.

5. Schuster, S.A., et al, "RAP. 2 - An Associative Processor for Data Bases and its Applications,"Proceedings 5th Annual Symposium on Computer Architecture, April 1978, pp. 52-59.

6. Baum, R.I., et al, "The Architecture of a Data Base Computer—Part 1: Concepts and Capabilities," Ohio State University Tech. Report No. OSU-CISRC-TR-76-1, Sept. 1976.

7. Hsiao, D.K., and K. Kannan, "The Architecture of a Database Computer—Part 2: The Design of Structure Memory and its Related Computers," Ohio State University Tech. Report No. OSU-CISRC-TR-76-2, Oct. 1976.

8. Hsiao, D.K., and K. Kannan, "The Architecture of a Data Base Computer—Part 3: The Design of the Mass Memory and its Related Components," Ohio State University Tech. Report No. OSU-CISRC-TR-76-3, Dec. 1976.

9. Severance, D. G., and J. V. Carlis, "A Practical Approach to Selecting Record Access Paths," Computing Surveys, Vol. 9, No. 4, Dec. 1977, pp. 259-272.

10. Chamberlin, D. D., & R. F. Boyce, "SEQUEL: A Structured English Query Language," Proceedings ACM Workshop on Data Description, Access, and Control, 1974, pp. 249-264.

11. Sperry Univac, Query Language Processor (QLP 1100), UP-8231 Rev. 1, 1977.

12. Ampex Corp., PTD-930x Parallel Transfer Drive, Product Description 3308829-01, Oct. 1978.

13. E.F. Codd, "Access Control for Relational Database System" B.C.S. Symposium, London April 5, 1973.

14. C.J. Date, An Introduction to Database Systems Addison Wesley (1977).

APPENDIX A. DATA MANAGEMENT QUESTIONAIRE

DATA MANAGEMENT QUESTIONNAIRE

A.1. Environment

1. Type of business
   (e.g., auto rental, telephone, manufacturing, ...) __________

2. Major data base applications of business
   (e.g., reservations, billing, directory assistance, ...) __________

	Application	% of total system activity	% of total data base activity	Data base name
a.	________	________ %	________ %	________
b.	________	________ %	________ %	________
c.	________	________ %	________ %	________
d.	________	________ %	________ %	________
e.	________	________ %	________ %	________

A.2. Application Usage and Description

1.	Name of Application to be described	________
		Transaction / Batch
2.	Application operation percentage	____ % / ____ %
3.	Quantity of records retrieved per transaction/execution	
	a. one b. few c. many d. all	______ / ______
4.	Response time required for above quantity retrieved	______ / ______
5.	Fields of retrieved record most frequently required	
	a. one b. multiple c. all	______ / ______
6.	When updating, quantity of records updated per transaction/execution	
	a. one b. few c. many d. all	______ / ______
7.	Percentage of transactions/executions requiring updates	____ % / ____ %
8.	Response time required for quantity updated	______ / ______
9.	Number of concurrent users	______ / ______
10.	Number of concurrent users allowed to update	______ / ______

11. For a transaction application, what are the number of transactions per hour

a. average	________ /
b. maximum	________ /

12. For this application, have all of the search keys been identified in advance	________ / ________
13. Maximum number of search keys used in application	________ / ________

14. What percent of the requests use

a. no search keys (sequential pass)	_____ % / _____ %
b. 1 search key	_____ % / _____ %
c. more than 1 search key	_____ % / _____ %

15. Number of levels or record types traversed to obtain selected record:

a. average number	________ / ________
b. maximum number	________ / ________

A.3. Data Base Usage and Description

1. Name of data base to be described ____________

2. Total size of data base ____________ bytes

3. Average growth rate of data base per month ____________ %

4. Number of fixed length record types ____________

5. Number of variable length record types ____________

6. Considering all of the record types, what is the record size:

   a. minimum ____________ bytes
   b. average ____________ bytes
   c. maximum ____________ bytes

7. Number of record types accessed through following methods:

   a. sequential ____________
   b. index sequential ____________
   c. random (DIRECT, CALC) ____________
   d. VIA (another record type) ____________

8. Organization of record types

   a. not related ____________
   b. hierarchy

b1. maximum number of levels ____________

b2. average number of levels ____________

c. network

c1. maximum number of sets to which a record type can belong ____________

c2. average number of sets to which a record type can belong ____________

9. Considering all of the record types, what is the number of search keys:

a. minimum ____________

b. average ____________

c. maximum ____________

10. What percent of the transactions change key field values in a record ____________ %

RESPONDEE	
ORGANIZATION	
LOCATION	
PHONE	

Briefly describe the application detailed in this questionnaire:

APPENDIX B. SUMMARY TABLES

DATA MANAGEMENT QUESTIONNAIRE SUMMARY

TOTAL QUESTIONNAIRES RECEIVED TO DATE: 42

B.1. Environment

1. Types of businesses:

   manufacturer, university, government, publisher, banking, telephone, airline, etc.

2. Major data base applications of businesses:

   parts inventory, student records, elections, subscriptions, on-line banking, directory assistance, reservations, etc.

	(minimum:mean:maximum)
% of total system activity	0.5:36:100
% of total data base activity	1:60:100

B.2. Application Usage and Description

		(minimum:mean:maximum) Transaction/ Batch
2.	Application operation percentage	0:52:100%/ 0:60:100
3.	Quantity of records retrieved per transaction/execution a. one b. few c. many d. all	a:b:d / b:c:d
4.	Response time required for above quantity retrieved	.15:5:30sec/ 1hour/overnight/overnig
5.	Fields of retrieved record most frequently required a. one b. multiple c. all	a:b:c / a:b:c
6.	When updating, quantity of records updated per transaction/execution a. one b. few c. many d. all	a:b:d / a:c:d
7.	Percentage of transactions/executions requiring updates	0:48:100%/ 0:52:100%
8.	Response time required for quantity updated	.2:5:30 / 1hour/overnight/overn
9.	Number of concurrent users	1:10:4000/ 1:3:15

10.	Number of concurrent users allowed to update	0:5:4000 / 1:2:5
11.	For a transaction application, what are the number of transactions per hour	
	a. average	2:400:100000
	b. maximum	10:1000:180000
12.	For this application, have all of the search keys been identified in advance	no:yes:yes / no:yes:yes
13.	Maximum number of search keys used in application	1:3:40 / 0:2:40
14.	What percent of the requests use	
	a. no search keys (sequential pass)	0:7:100% / 0:25:100%
	b. 1 search key	0:58:100%/ 0:36:100%
	c. more than 1 search key	0:35:100%/ 0:37:100%
15.	Number of levels or record types traversed to obtain selected record:	
	a. average number	0:2:8 / 1:2.5:8
	b. maximum number	1:5:16 / 1:5:16

B.3. Data Base Usage and Description

		(minimum:mean:maximum)
2.	Total size of data base	20KB:600MB:24000MBytes
3.	Average growth rate of data base per month	0:2.6:10%
4.	Number of fixed length record types	0:23:628
5.	Number of variable length record types	0:0:80
6.	Considering all of the record types, what is the record size:	
	a. minimum	1:20:640 bytes
	b. average	30:347:5300 bytes
	c. maximum	126:1493:10752 bytes
7.	Number of record types accessed through following methods:	
	a. sequential	0:4:43
	b. index sequential	0:3:29
	c. random (DIRECT, CALC)	0:28:628
	d. VIA (another record type)	0:21:84
8.	Organization of record types	
	b. hierarchy	
	b1. maximum number of levels	1:4:16
	b2. average number of ievels	1:2:8

c. network

c1. maximum number of sets to which a record type can belong	1:3:5
c2. average number of sets to which a record type can belong	1:1:15

9. Considering all of the record types, what is the number of search keys:

a. minimum	0:1:10
b. average	1:3:12
c. maximum	1:6:32

10. What percent of the transactions change key field values in a record	0:.5:21%